Bounded-Concurrent Secure Two-Party Computation Without Setup Assumptions*

Yehuda Lindell

IBM T.J.Watson Research
19 Skyline Drive, Hawthorne
New York 10532, USA
lindell@us.ibm.com

May 23, 2003

Abstract

In this paper we study the feasibility of obtaining protocols for general two-party computation that remain secure under concurrent composition. (A general protocol can be used for obtaining secure computation of any functionality.) We consider a scenario where *no trusted setup* is assumed (and so, for example, there is no common reference string available to the parties); this is also called the "plain model". We present both negative and positive results for this model. Specifically, we show that a general two-party protocol that remains secure for m concurrent executions and can be proven via *black-box* simulation, must have more than m rounds of communication. An important corollary of this result is that *there do not exist* protocols for black-box secure general two-party computation for the case of unbounded concurrency (where any polynomial number of concurrent executions may be run). On the positive side, we show that under general cryptographic assumptions, there exist secure protocols for general two-party computation in the model of bounded concurrent composition (in this model the number of concurrent executions is fixed and the protocol design may depend on this number). Our protocol has $O(m)$ rounds of communication, where m is the bound on the number of concurrent executions, and uses both black-box and non black-box techniques. We note that this protocol constitutes the first feasibility result for general two-party computation without setup assumptions *for any model of concurrency.*

Keywords: secure two-party computation, concurrent composition, black-box and non black-box simulation.

*An extended abstract of this paper appeared in the 35th STOC, 2003. This is a *preliminary* full version.

Contents

1 Introduction **1**

2 Definitions: m-Bounded Concurrent Secure Computation **5**

3 Lower Bounds **8**
3.1 The Main Result 8
3.2 Impossibility For Concurrent Oblivious Transfer 17
3.3 Extensions of the Lower Bounds 18

4 Tools for our Protocol – Zero-Knowledge **19**
4.1 The Zero-Knowledge Proof of Knowledge of [4] 19
4.2 Black-Box Bounded Concurrent ZK 21
4.3 Non-Black-Box Bounded Concurrent ZK 27

5 A Special-Purpose Composition Theorem **30**
5.1 The ZK-Hybrid Model 30
5.2 Motivation for the Composition Theorem 31
5.3 The Composition Theorem 32
5.4 Generalizing the Composition Theorem 42

6 Obtaining Bounded Concurrent Two-Party Computation **42**

References **44**

A Blum's Protocol for Hamiltonicity [6] **47**

1 Introduction

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). These security requirements must hold in the face of a malicious adversary who controls one of the parties and can arbitrarily deviate from the protocol instructions (i.e., in this work we consider static adversaries). Powerful feasibility results have been shown for this problem, demonstrating that *any* two-party probabilistic polynomial-time functionality can be securely computed, assuming the existence of trapdoor permutations [40, 25].

Security under concurrent composition. The above-described feasibility results relate only to the stand-alone setting, where a single pair of parties run a single execution. A more general (and realistic) setting relates to the case that many protocol executions are run concurrently within a network. Unfortunately, the security of a protocol in the stand-alone setting does not necessarily imply its security under concurrent composition. Therefore, an important research goal is to re-establish the feasibility results of the stand-alone setting for the setting of concurrent composition.

We stress that the need for security under concurrent composition is not merely an issue of efficiency. Specifically, one cannot claim that in order to ensure security, all executions should just be carried out sequentially (recall that security under sequential composition is guaranteed by the stand-alone definitions [7]). This is because the honest parties must actively coordinate their executions in order to ensure sequentiality. However, the honest parties may not even know of each other's existence. Therefore, they cannot coordinate their executions, and a dishonest adversary that interacts with a number of parties can schedule the executions concurrently. The protocol being used must therefore be secure under concurrent composition.

The study of security under concurrent composition was initiated in the context of concurrent zero-knowledge [16, 14]. In this scenario, a prover runs many copies of a protocol with many verifiers, and zero-knowledge must be preserved. The feasibility of concurrent zero-knowledge in the *plain model* (where no trusted setup or preprocessing phase is assumed) was first demonstrated by [37]. In general, the issue of concurrent zero-knowledge has received much attention, resulting in a rather exact understanding of the round-complexity of black-box concurrent zero-knowledge [30, 39, 10, 29, 35]. Another specific problem that has been studied in the context of concurrent composition is that of oblivious transfer [18].

In this paper, we consider the question of concurrent composition for *general* secure two-party computation. (A general protocol is one that can be used for obtaining secure computation of any efficient functionality.) The problem of concurrent composition can be defined in many different ways. One possibility is the model used in the framework of universal composability [8]. This framework considers a very general setting where many sets of (possibly different) parties run many protocols, and secure protocols may run concurrently with arbitrary other protocols. The problem of universally composable two-party computation has been studied with the following results. It is possible to securely compute *any* two-party functionality in a universally composable way, in the *common reference string model*[1] [12]. In contrast, in the plain model, the possibility of obtaining universally composable two-party computation has been ruled out for a very large class of functionalities [9, 8, 11]. However, these impossibility results relate specifically to the

[1] In this model, all parties are given access to a string that is ideally chosen from some distribution.

definition of universal composability and not necessarily to the composition operation guaranteed to hold under this definition. Thus it may be possible to obtain concurrent secure computation for other definitions and/or other models of concurrency. One specific model of interest is the model considered for concurrent zero-knowledge, where two parties run the same protocol many times concurrently. We remark that apart from the work on universal composability, there has been no work on the feasibility of concurrent secure two-party computation in the plain model (for any model of concurrency).

Thus, one could loosely sum up our current understanding of concurrent composition as follows. In the common reference string model, the feasibility of universally composable two-party computation has been demonstrated. However, in the plain model, almost all known results relate specifically to concurrent zero-knowledge. In particular, *nothing* is known about the feasibility of obtaining concurrent general secure two-party computation in the plain model for definitions other than those of universal composability. Thus, a fundamental question that remains unanswered is the following:

> *Can general secure two-party computation that composes concurrently be obtained in the plain model?*

This question is of both theoretical interest and practical importance. (We note that in many settings, it is not clear how to reasonably obtain the trusted setup of a common reference string.) The aim of this work is take a first step in providing an answer to this question.

The model of concurrency. The notion of concurrent composition relates to a large number of possible scenarios. These scenarios range from two parties running the same protocol many times concurrently, to many pairs of different parties running many arbitrary protocols concurrently. In this work we consider the first case of two parties running the same protocol many times concurrently. We note that this is actually equivalent to the case where many different pairs of parties run a protocol concurrently, with the following adversarial limitation. Each party is designated to be either a first party P_1 or a second party P_2 (this defines the role that it plays in the protocol execution). The adversary is then only allowed to corrupt a subset of the parties playing P_1 or a subset of the parties playing P_2; but cannot simultaneously corrupt parties playing P_1 and parties playing P_2. Such a scenario can model some real-world scenarios, like a server concurrently interacting with many clients (and where we assume that there is no simultaneous corruption of a server and a client). From here on, when we talk of "concurrent composition", we mean this model. We remark that this is the exact model considered in the entire body of work on concurrent zero-knowledge. Although this is a rather limited model, we believe that it serves as a good first step to understanding concurrent composition of secure protocols. Furthermore, we stress that even in this limited model, nothing is currently known when no trusted setup is assumed.

An important parameter which must be considered when defining the model is the "amount of concurrency" that is allowed. In most of the literature on concurrent zero-knowledge, security must hold for *any* polynomial number of executions. We call this model unbounded concurrency. In contrast, [1] considered a model where the number of concurrent executions is a fixed polynomial in the security parameter, and the protocol design can depend on this number. This model is called bounded concurrency, and when m is the maximum number of concurrent executions, we talk about m-bounded concurrent composition ($m = m(n)$ is a fixed polynomial in the security parameter). Our protocol for general secure two-party computation is cast in the setting of bounded concurrent composition.

Our Results

As we have described, in this work we consider the case of a single pair of parties running the same protocol, in the setting of m-bounded concurrent composition. The definition of security that we use is a natural extension of the ideal-model based definitions of [26, 33, 5, 7] for the stand-alone setting. We present both negative and positive results relating to the feasibility of (bounded) concurrent secure two-party computation in the plain model, under this definition. We begin by stating our negative result, which is actually a black-box lower bound:

Negative results (black-box lower bound): We show that secure two-party computation with black-box simulation[2] is "hard" to achieve, even in the model of bounded concurrency:

Theorem 1.1 (lower bound): *There exists an efficient functionality f such that every protocol Π that securely computes f under m-bounded concurrent composition, and is proven using* black-box simulation*, must have more than m rounds of communication.*

We remark that the specific functionality referred to in the theorem is a natural one. Specifically, we show our lower bound for blind signatures [13], and 1-out-of-2 oblivious transfer [36, 15]. This result seemingly contradicts the fact that protocols for concurrent oblivious transfer have been demonstrated [18]. However, in the model of [18], all the inputs in all the executions are independent of each other. In contrast, we consider a more standard model where quantification is over all inputs, and in particular, over possibly correlated inputs.

The proof of Theorem 1.1 works by showing that when concurrent composition is considered, the "rewinding capability" of the simulator is severely limited. In fact, for a protocol of m rounds that is run m times concurrently, there exists a scheduling of messages so that in one of the executions, the simulator is unable to carry out any rewinding of the adversary. However, informally speaking, a black-box simulator must rewind in order to successfully simulate.

Theorem 1.1 implies two important corollaries. First, it demonstrates a black-box lower bound for *general* two-party computation (i.e., protocols that can be used for obtaining secure computation of any efficient functionality). Second, it shows that black-box secure protocols for the unbounded concurrent model *do not exist*. This is because the number of rounds of a protocol must be bounded by a fixed polynomial. However, Theorem 1.1 states that when the number of concurrent executions is greater than this number of rounds (as can happen in the model of unbounded concurrency), security cannot hold. In summary, we have the following corollary:

Corollary 1.2 *There do not exist secure protocols for general two-party computation in the unbounded concurrent model, that can be proven using black-box simulation.*

This corollary stands in stark contrast to the setting of concurrent zero-knowledge, where black-box simulation does suffice for obtaining unbounded concurrent composition [37]. Thus, obtaining concurrent secure computation is strictly harder than obtaining concurrent zero-knowledge. This result also highlights a limitation on concurrent zero-knowledge. Specifically, the model for concurrent zero-knowledge considers the case that the zero-knowledge protocol is run concurrently with itself only. However, arguably the most important application of zero-knowledge is its use as a

[2] In the setting of secure computation, security is proven by demonstrating that for every real-model adversary $\mathcal{A}$ there exists an ideal-model adversary/simulator $\mathcal{S}$ such that the output distribution generated by $\mathcal{S}$ in an ideal execution is indistinguishable from that generated by $\mathcal{A}$ in a real execution of the protocol; see Definition 1. We say that an ideal-model simulator $\mathcal{S}$ is *black-box* if it is given only oracle access to the real-model adversary $\mathcal{A}$.

building block in larger secure protocols. Thus, one would hope that concurrent zero-knowledge could be "plugged-in" to a larger protocol, in order to obtain concurrent security for the larger protocol. Our lower bound proves that this cannot be done in a generic way.[3] Despite this, our upper bound *does* use concurrent zero-knowledge protocols and techniques (in a non-generic way) in order to achieve bounded concurrent secure two-party computation; see below.

We note that Theorem 1.1 holds even if at any given time, at most *two* executions are running simultaneously. (Loosely speaking, in such a case the m-bounded concurrency means that m different protocol executions may overlap.) This shows that the lower bound does not stem from deep protocol nesting as in the case in concurrent zero-knowledge. Indeed, a nesting depth of at most two is needed here.

Positive results (feasibility): Our positive result states that any efficient two-party functionality can be securely computed under bounded concurrent secure computation:

Theorem 1.3 (upper bound): *Assume that enhanced trapdoor permutations*[4] *and collision resistant hash functions exist. Then, for any two party functionality f and for any m, there exists a protocol Π that securely computes f under m-bounded concurrent composition.*

We remark that Theorem 1.3 is the *first* feasibility result for general two-party computation in the plain model, *for any model of concurrency.*

We now provide a very brief and informal outline of the proof of the theorem. We begin by observing that secure two-party computation that composes concurrently can be obtained in a hybrid model where the parties have (concurrent) access to a trusted party computing the ideal zero-knowledge functionality. This was formally demonstrated in [12]. Given this observation, the question is whether or not one can transform protocols that use this ideal zero-knowledge functionality into protocols that run in the real model (without any trusted help). Of course, this transformation must preserve the concurrent composition, or bounded concurrent composition, of the protocol.

A naive implementation of the above idea is to replace the ideal zero-knowledge calls with any concurrent zero-knowledge protocol. However, two problems arise with this idea. First, as we have mentioned, concurrent zero-knowledge considers a scenario where the protocol is run concurrently with itself only. Thus, it is not clear that the zero-knowledge simulation can be accomplished if the protocol is run concurrently to other protocols as well. Second, a problem relating to the malleability of protocols arises. That is, during the concurrent executions, the adversary may simultaneously verify and prove zero-knowledge proofs. Thus, it can execute a man-in-the-middle attack on the zero-knowledge proofs, possibly enabling it to prove false statements. This is a problem because in order for the simulation of the secure protocol to work, we must be sure that the proofs provided by the adversary are sound. This second problem is solved by having the parties use fundamentally different zero-knowledge proofs.[5] Specifically, one party proves using the black-box zero-knowledge

[3] To explain this more rigorously, we note that black-box secure protocols that use ideal calls to zero-knowledge and compose concurrently exist [12]. Thus, one would hope that a concurrent zero-knowledge protocol could be used to replace such an ideal call and the resulting protocol would remain secure. However, when the zero-knowledge protocol is black-box zero-knowledge, such a strategy would result in a black-box secure protocol that composes concurrently, in contradiction to Corollary 1.2.

[4] Enhanced trapdoor permutations have the property that a random element generated by the domain sampler is hard to invert, even given the random coins used by the sampler; see [21, Appendix C]. We note that the construction of [25] for secure two-party computation in the stand-alone model assumes the existence of enhanced trapdoor permutations.

[5] We note that the recent result of [2] for "non-malleable coin-tossing" does not solve this problem. This is because [2] relies on a very specific scheduling which can be enforced in their scenario, but cannot be enforced here.

protocol of [35, 37], while the other party proves using the non black-box zero-knowledge protocol of [1, 3]. It turns out that a careful choice of parameters for these protocols solves the problem of malleability. However, the first problem (of concurrency with other protocols) still remains. This is solved as follows. The protocol of [1] can easily be modified so that it remains zero-knowledge when run concurrently with other protocols (intuitively, this is the case because there is no rewinding). However, the protocol of [35] is more problematic. We therefore devise a *new* black-box simulation strategy for [35] in the setting of m-bounded concurrency, that enables it to be used directly as a sub-protocol of another protocol.

Having solved these problems, we are able to replace the ideal zero-knowledge calls in any protocol with the specific m-bounded concurrent zero-knowledge protocols described above. Putting this transformation together with a protocol that is secure when given access to the ideal zero-knowledge functionality, we obtain a protocol that is secure under m-bounded concurrent composition in the real model. We note that our protocol has $O(m)$ rounds and uses both black-box and non black-box techniques (thus, our upper bound does not "match" the lower bound).

We conclude with a remark about *efficiency*. Our protocol requires $O(m)$ rounds to obtain security for m concurrent executions. It is therefore far from being "reasonably efficient". Nevertheless, the focus of this paper is *not* efficiency. Rather, we believe that establishing the feasibility of obtaining secure computation in a reasonable model of concurrency is of great importance, irrespective of efficiency.

2 Definitions: m-Bounded Concurrent Secure Computation

In this section we present the definition for m-bounded concurrent secure two-party computation. The basic description and definition of secure computation follows [26, 33, 5, 7]. We denote computational indistinguishability by $\stackrel{c}{\equiv}$, and the security parameter (and, for simplicity, the lengths of the parties' inputs) by n.

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 $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, 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 $f_2(x, y)$. We often denote such a functionality by $(x, y) \mapsto (f_1(x, y), f_2(x, y))$. Thus, for example, the zero-knowledge proof of knowledge functionality for a relation R, is denoted by $((x, w), x) \mapsto (\lambda, R(x, w))$. In the context of concurrent composition, each party actually receives a vector of inputs of polynomial length, and the aim of the parties is to jointly compute $f(x_i, y_i)$ for every i. The fact that m-bounded concurrency is considered relates to the allowed scheduling of messages by the adversary in the protocol executions; see the description of the real model below.

We note that our results here also apply to *reactive functionalities* where inputs and outputs are supplied over a number of stages. Such a functionality can be modeled by a probabilistic polynomial-time interactive machine who receives inputs and supplies outputs. This machine can keep state and thus the inputs from previous computations can influence the outputs of later ones.

Adversarial behavior. In this work we consider a malicious, static adversary. That is, the adversary controls one of the parties (who is called corrupted) and may then interact with the honest party while arbitrarily deviating from the specified protocol. The focus of this work is not

on fairness. We therefore present a definition where the adversary always receives its own output and can then decide when (if at all) the honest party will receive its output. The scheduling of message delivery is decided by the adversary.

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 trivially secure. 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. Unlike in the case of stand-alone computation, here the trusted party computes the functionality m times, each time upon different inputs. 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.

Concurrent execution in the ideal model. Let $p(n)$ be a polynomial in the security parameter. Then, an ideal execution with an adversary who controls either P_1 or P_2 proceeds as follows:

Inputs: The honest party and adversary each obtain a vector of $p(n)$ inputs of length n; denote this vector by $\overline{w}$ (i.e., $\overline{w} = \overline{x}$ or $\overline{w} = \overline{y}$).

Honest party sends inputs to trusted party: The honest party sends its entire input vector $\overline{w}$ to the trusted party.

Adversary interacts with trusted party: For every $i = 1, \ldots, p(n)$, the adversary can send (i, w'_i) to the trusted party, for any $w'_i \in \{0,1\}^n$ of its choice. Upon sending this pair, it receives back its output based on w'_i and the input sent by the honest party. (That is, if P_1 is corrupted, then the adversary receives $f_1(w'_i, y_i)$ and if P_2 is corrupted then it receives $f_2(x_i, w'_i)$.) The adversary can send the (i, w'_i) pairs in any order it wishes and can also send them *adaptively* (i.e., choosing inputs based on previous outputs). The only limitation is that for any i, *at most one pair* indexed by i can be sent to the trusted party.

Trusted party answers honest party: Having received all of its own outputs, the adversary specifies which outputs the honest party receives. That is, the adversary sends the trusted party a set $I \subseteq \{1, \ldots, p(n)\}$. Then, the trusted party supplies the honest party with a vector $\overline{v}$ of length $p(n)$ such that for every $i \notin I$, $v_i = \perp$ and for every $i \in I$, v_i is the party's output from the i^{th} execution. (That is, if P_1 is honest, then for every $i \in I$, $v_i = f_1(x_i, w'_i)$ and if P_2 is honest, then $v_i = f_2(w'_i, y_i)$.)

Outputs: The honest party always outputs the vector $\overline{v}$ that it obtained from the trusted party. The adversary may output an arbitrary (probabilistic polynomial-time computable) function of its initial input and the messages 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 $\mathcal{S}$ be a non-uniform probabilistic expected polynomial-time machine (representing the ideal-model adversary). Then, the ideal execution of f (on input vectors $(\overline{x}, \overline{y})$ of length $p(n)$ and auxiliary input z to $\mathcal{S}$), denoted $\text{IDEAL}_{f,\mathcal{S}}(\overline{x}, \overline{y}, z)$, is defined as the output pair of the honest party and $\mathcal{S}$ from the above ideal execution.

We note that the definition of the ideal model does not include any reference to the bound m on the concurrency. This is because this bound is relevant only to the scheduling allowed to the adversary in the real model; see below. However, the fact that a concurrent setting is considered can be seen from the above-described interaction of the adversary with the trusted party. Specifically, the adversary is allowed to obtain outputs in any order that it wishes, and can choose its inputs adaptively based on previous outputs. This is inevitable in a concurrent setting where the adversary can schedule the order in which all protocol executions take place. (In particular, the adversary can schedule the executions sequentially, thereby learning previous outputs before defining the next input.)

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). Let $p(n)$ and $m = m(n)$ be polynomials, let f be as above and let Π be a two-party protocol for computing f. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine that controls either P_1 or P_2. Then, the real m-bounded concurrent execution of Π (on input vectors $(\overline{x}, \overline{y})$ of length $p(n)$ and auxiliary input z to $\mathcal{A}$), denoted $\text{REAL}^m_{\Pi,\mathcal{A}}(\overline{x}, \overline{y}, z)$, is defined as the output pair of the honest party and $\mathcal{A}$, resulting from $p(n)$ executions of the protocol interaction, where the honest party always inputs its i^{th} input into the i^{th} execution.

The scheduling of all messages throughout the executions is controlled by the adversary. That is, the execution proceeds as follows. The adversary sends a message of the form (i, α) to the honest party. The honest party then adds α to the view of its i^{th} execution of Π and replies according to the instructions of Π and this view.[6] The adversary continues by sending another message (j, β), and so on. When unbounded concurrency is considered, *any* scheduling of the messages by the adversary is allowed. In contrast, in the setting of m-bounded concurrency, the scheduling by the adversary must fulfill the following condition: for every execution i, from the time that the i^{th} execution begins until the time that it ends, messages from at most m different executions can be sent. (Formally, view the schedule as the ordered series of messages of the form (index, message) that are sent by the adversary. Then, in the interval between the beginning and termination of any given execution, the number of different indices viewed can be at most m.) We note that this definition of concurrency covers the case that m executions are run simultaneously. However, it also includes a more general case where many more than m executions take place, but each execution overlaps with at most m other executions.

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 for every real-model adversary there exists an ideal model adversary that can simulate an execution of the secure real-model protocol.

Definition 1 (m-bounded concurrent secure computation): *Let $m = m(n)$ be a polynomial and let f and Π be as above. Protocol Π is said to* securely compute *f* under *m-bounded concurrent composition if for every real-model non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ controlling party P_i for $i \in \{1, 2\}$, there exists an ideal-model non-uniform probabilistic expected polynomial-time*

[6] Notice that the honest party runs each execution of Π obliviously to the other executions. Thus, this is stateless composition.

adversary $\mathcal{S}$ controlling P_i, such that for every polynomial $p(n)$,

$$\left\{\text{IDEAL}_{f,\mathcal{S}}(\overline{x},\overline{y},z)\right\}_{n\in\mathsf{N};\overline{x},\overline{y}\in(\{0,1\}^n)^{p(n)};z\in\{0,1\}^*} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}^m_{\Pi,\mathcal{A}}(\overline{x},\overline{y},z)\right\}_{n\in\mathsf{N};\overline{x},\overline{y}\in(\{0,1\}^n)^{p(n)};z\in\{0,1\}^*}$$

Definition 1 follows a liberal interpretation of "efficient simulation" in that it allows the ideal-model simulator/adversary to run in *expected* polynomial-time. Formally, an interactive machine M_1 is expected polynomial-time if there exists a (single) polynomial $q(\cdot)$ 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 $q(n)$.

Black-box security. We say that a protocol Π for securely computing f is proven using black-box simulation if $\mathcal{S}$ is given only oracle access to $\mathcal{A}$. That is, there exists an ideal-model adversary $\mathcal{S}$ such that for every real-model adversary $\mathcal{A}$,

$$\left\{\text{IDEAL}_{f,\mathcal{S}^{\mathcal{A}(z)}}(\overline{x},\overline{y},z)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}^m_{\Pi,\mathcal{A}}(\overline{x},\overline{y},z)\right\}$$

We stress that in this case, $\mathcal{S}$ is not given the auxiliary input z, but rather has oracle access to $\mathcal{A}(z)$ only.[7]

Remark. Definition 1 is rather simplistic in that it assumes that all inputs are fixed before any execution begins. We remark that our upper bound also holds for a more general definition where inputs for later executions can depend on outputs of earlier executions.

3 Lower Bounds

In this section we prove our black-box lower bounds for m-bounded concurrent secure two-party computation. Specifically, we show the existence of an efficient functionality f, such that every protocol that securely computes f under m-bounded concurrent composition, and is proven using *black-box* simulation, must have more than m rounds of communication. In order to simplify the exposition, we count a round of communication to be a pair of messages sent between the parties. Thus, in more common terminology, we actually show that more than $2m$ rounds are required for obtaining m-bounded concurrent composition.

3.1 The Main Result

As described above, two-party protocols are proven secure by demonstrating that for every real model adversary $\mathcal{A}$ controlling one of the parties, there exists an ideal model adversary/simulator $\mathcal{S}$ who can simulate a real execution for $\mathcal{A}$. The adversary $\mathcal{S}$ interacts with the trusted third party to whom it sends input and receives output. Typically, $\mathcal{S}$ extracts an input used by $\mathcal{A}$ and sends this to the trusted party. The trusted party then computes the output (based on the input sent by $\mathcal{S}$ and the input of the honest party) and returns the output to $\mathcal{S}$. Having received this output, $\mathcal{S}$ completes the simulation, ensuring that $\mathcal{A}$'s view is consistent with the output received from the trusted party. Since the output of $\mathcal{A}$ depends on the input of the honest party, $\mathcal{S}$ must

[7] The proof of our lower bound (Theorem 1.1) relies on the fact that the black-box simulator $\mathcal{S}$ is not given the adversary's auxiliary input. Withholding the auxiliary input from $\mathcal{S}$ is standard practice, and is used in an essential way in all black-box lower bounds for zero-knowledge. Indeed, most known (zero-knowledge) black-box lower bounds are known to *not* hold when the simulator receives the adversary's auxiliary input [1].

query the trusted party in order to complete the simulation. The fact that $\mathcal{S}$ needs to interact with the trusted party is crucial to the lower bound. This is fundamentally different to the case of zero-knowledge where the verifier's output is always the same (i.e., a bit saying accept). Thus a simulator for zero-knowledge need not have any interaction with an external trusted third party. This difference explains why our lower bound does not apply to concurrent zero-knowledge.

Continuing the above idea, it follows that for $\mathcal{S}$ to simulate m executions of a secure protocol, it must query the trusted party m times. Another key observation is that once the input of an execution has been sent to the trusted party, further rewinding of $\mathcal{A}$ is problematic. This is because $\mathcal{A}$ may choose its inputs depending on the messages it has seen. Therefore, rewinding $\mathcal{A}$ may result in $\mathcal{A}$ modifying its input. Since this modified input would also need to be sent to the trusted party and only one input can be sent to the trusted party in each execution, it is not possible to rewind $\mathcal{A}$ in an effective way.

Next, consider the following scheduling of messages in a concurrent execution: Between every round of messages in the first execution, place a complete protocol execution. We remark that this scheduling is possible if the protocol has m rounds and remains secure for m concurrent executions. (Also, notice that at any given time, at most two different protocol executions are actually running simultaneously.) Then, by the above discussion, $\mathcal{S}$ must send an input to the trusted party between every round of the first execution (otherwise $\mathcal{S}$ can simulate a complete execution without conversing with the trusted party). Since $\mathcal{S}$ cannot rewind $\mathcal{A}$ behind the points at which input is sent to the trusted party, this implies that $\mathcal{S}$ cannot rewind $\mathcal{A}$ at all in the first execution. However, $\mathcal{S}$ is a black-box simulator and it must be able to rewind in order to successfully simulate. We conclude that one cannot obtain a secure protocol that has m rounds and remains secure for m concurrent executions. We now proceed to the formal proof.

Theorem 2 *There exists a probabilistic polynomial-time functionality f such that every protocol that securely computes f under m-bounded concurrent composition and is proven using* black-box simulation*, must have more than m rounds of communication.*

Proof: We prove this theorem under the assumption that one-way functions exist. This suffices because it has been shown that secure coin tossing (even for a single execution) implies the existence of one-way functions [28]. Therefore, if one-way functions do not exist, the coin tossing functionality already fulfills the requirement of the theorem statement.

Our proof is based on the motivating discussion above and is according to the following outline. We begin by defining a functionality f for which the lower bound holds, and assume by contradiction that there exists an m-round protocol Π that securely computes f under m-bounded concurrent composition. Next, a specific real-model adversary $\mathcal{A}$ is constructed who controls P_2 and interacts with P_1 in an execution of Π. By the security of Π, there exists an ideal-model simulator $\mathcal{S}$ for $\mathcal{A}$. The adversary $\mathcal{A}$ is constructed so that we can claim certain properties of the simulator $\mathcal{S}$. One important claim about $\mathcal{S}$ will be that in one of the executions, it is essentially unable to rewind $\mathcal{A}$. Furthermore, in this execution, $\mathcal{A}$ basically plays the role of an honest P_2. Therefore, we have that $\mathcal{S}$ can actually "simulate" while interacting with an honest P_2, whom it cannot rewind. This observation is used to construct a new adversary $\mathcal{A}'$ who controls P_1 and attacks P_2. In this attack, $\mathcal{A}'$ invokes $\mathcal{S}$ who proceeds to "simulate" for P_2. The proof is then concluded by showing that this enables $\mathcal{A}'$ to obtain information that cannot be obtained in the ideal-model, contradicting the security of Π.

As mentioned in the above outline, we begin by defining a functionality f for which the lower bound holds. The functionality definition refers to a secure signature scheme [27] which can be constructed from any one-way function [38]. It actually suffices for us to use a *one-time* signature

scheme. Before defining the functionality, we present an informal definition of secure one-time signature schemes. A signature scheme is a triplet of algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, where Gen is a probabilistic generator that outputs a pair of signing and verification keys (sk, vk), Sign is a signing algorithm and Verify is a verification algorithm. Without loss of generality, we assume that the signing key sk explicitly contains the verification key vk. The validity requirement for signature schemes states that except with negligible probability, honestly generated signatures are almost always accepted; i.e., for almost every $(vk, sk) \leftarrow \mathsf{Gen}(1^n)$ and for every message x, $\mathsf{Verify}_{vk}(x, \mathsf{Sign}_{sk}(x)) = 1$. The security requirement of a signature scheme states that the probability that an efficient forging algorithm M can succeed in generating a valid forgery is negligible. This should hold even when M is given a signature to any single message of its choice. That is, in order for M to succeed, it must generate a valid signature on any message other than the one for which it received a signature. More formally, the following experiment is defined: The generator Gen is run, outputting a key-pair (vk, sk). Next, M is given vk and outputs a message x. Finally, M is given a signature $\sigma = \mathsf{Sign}_{sk}(x)$ and outputs a pair (x^*, σ^*). Then, we say that M succeeded if $\mathsf{Verify}_{vk}(x^*, \sigma^*) = 1$ and $x^* \neq x$. (That is, M output a valid signature on a message other than x.) We say that a one-time signature scheme is secure if for every non-uniform polynomial-time M, the probability that M succeeds is negligible.

We are now ready to define the functionality f, which can be seen as a "blind signature" type functionality [13]:

The functionality f:

- Inputs:
  - P_1 has a signing-key sk (for a secure one-time signature scheme).
  - P_2 has a verification-key vk and an input $\alpha \in \{0,1\}^n$.
- Output:
  - P_1 receives no output and P_2 receives $\sigma = \mathsf{Sign}_{sk}(\alpha)$.

That is, the functionality can be defined as follows:

$$(sk, (vk, \alpha)) \mapsto (\lambda, \mathsf{Sign}_{sk}(\alpha))$$

There are three important features of this functionality. First, in the ideal model, in order to compute the output of P_2, the trusted third party must be queried. (Otherwise, a signature would be generated without access to sk and this would constitute a forgery.) Second, if the verification-key vk given to P_2 is the key associated with sk, then party P_2 can check by itself that it received the correct output. Combined with the first observation, this means that P_2 can essentially check whether or not the trusted party was queried in an ideal execution. Third, P_1 learns nothing about P_2's input. These features are central in the proof of the lower bound.

We note that secure computation requirements are for all inputs and all sufficiently large n's. Therefore, the security of a protocol must also hold when P_1 and P_2 are given randomly generated associated signing and verification keys (i.e., vk and sk such that $(vk, sk) \leftarrow \mathsf{Gen}(1^n)$). From here on, we assume that the inputs of the parties P_1 and P_2 are always such that they have associated keys. We also assume that P_2's input α and $\mathcal{A}$'s auxiliary input z are uniformly distributed in $\{0,1\}^n$. (Once again, since security holds for all inputs, it also holds for uniformly distributed inputs.)

Proving the lower bound. In the remainder of the proof, we show that any m-bounded concurrent secure two-party protocol for computing f must have more than m rounds. By contradiction, assume that there exists such a protocol Π with exactly m rounds (if it has less, then empty messages can just be added). Without loss of generality, assume also that the first message of Π is sent by P_2.

We consider a scenario where $p(n) = m(n)$ (see Definition 1) and thus there are exactly m executions overall; denote these m executions of Π by $\Pi_1, \ldots, \Pi_m$ and denote the i^{th} round of execution Π_j by $\Pi_j(i)$. Also, denote the *messages* sent by P_1 and P_2 in $\Pi_j(i)$ by $\Pi_j(i)_1$ and $\Pi_j(i)_2$, respectively. Finally, denote the inputs of P_1 and P_2 in the i^{th} execution by sk_i and (vk_i, α_i), respectively, where each (vk_i, sk_i) is independently generated of all other pairs (vk_j, sk_j).[8] We now define an adversary $\mathcal{A}$ for Π.

The adversary $\mathcal{A}$: Adversary $\mathcal{A}$ corrupts party P_2 and schedules the m executions as follows. $\mathcal{A}$ plays $\Pi_1(1)$ (i.e., the first round of Π_1) and then runs the entire execution of Π_2. Next, $\Pi_1(2)$ is run and then the entire execution of Π_3. In general, the $(i-1)^{\text{th}}$ round of Π_1 is run followed by the entire execution of Π_i. Since there are m rounds in Π_1 and m concurrent executions, we have that between every two rounds $\Pi_1(i-1)$ and $\Pi_1(i)$ of Π_1, there is a complete execution of Π_i. See Figure 1 for a diagram of the schedule.

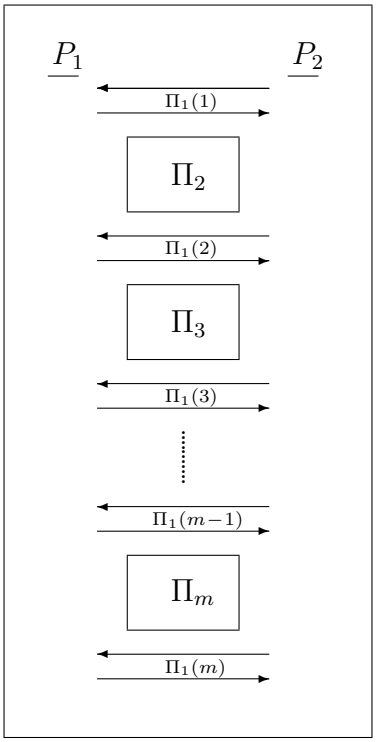

Figure 1: The schedule of the adversary. The arrows refer to messages from execution Π_1.

We now describe $\mathcal{A}$'s strategy within each execution. For the first execution Π_1, adversary $\mathcal{A}$ replaces its input (vk_1, α_1) with (vk_1, z); recall that z is $\mathcal{A}$'s auxiliary input and that it is uniformly distributed in $\{0,1\}^n$. Then, $\mathcal{A}$ runs execution Π_1 on this input, following the protocol

[8] We could have used the same pair (sk, vk) in all m executions and define the functionality f using a many-time signature scheme.

instructions exactly. So, apart from replacing part of its input, $\mathcal{A}$ plays exactly like the honest P_2 in execution Π_1.

For all other executions Π_i, adversary $\mathcal{A}$ behaves as follows. Recall that execution Π_i follows immediately after round $\Pi_1(i-1)$ (i.e., immediately after $\mathcal{A}$ receives message $\Pi_1(i-1)_1$). Now, $\mathcal{A}$ takes the input (vk_i, α_i) of P_2 and replaces α_i with $\Pi_1(i-1)_1$. Actually, the input α_i to f must be of length n. Therefore, $\mathcal{A}$ uses a pseudorandom function F [22] with a random key k, and replaces α_i with $F_k(\Pi_1(i-1)_1)$. Formally, the key k must be randomly chosen and included in $\mathcal{A}$'s auxiliary input; we omit this in order to simplify notation. (Recall that pseudorandom functions can be constructed from any one-way function, as needed.) After replacing α_i with $F_k(\Pi_1(i-1)_1)$, adversary $\mathcal{A}$ honestly follows the protocol instructions of P_2 in Π_i. By the definition of f, $\mathcal{A}$'s output from Π_i should be a value σ_i that constitutes a valid signature on $F_k(\Pi_1(i-1)_1)$ using signing-key sk_i (i.e., σ_i is such that $\mathsf{Verify}_{vk_i}(\sigma_i, F_k(\Pi_1(i-1)_1)) = 1$). Now, $\mathcal{A}$ checks that this holds for σ_i. If yes, $\mathcal{A}$ proceeds as in the above-described schedule. Otherwise, $\mathcal{A}$ sends $\perp$, outputs the output values from already completed executions (i.e., $\sigma_2, \ldots, \sigma_{i-1}$) and aborts. At the conclusion, $\mathcal{A}$ outputs all of its inputs and outputs from the protocol executions.

(Before proceeding with the proof, we provide some motivation to this construction of the adversary $\mathcal{A}$. The basic idea is to prevent any simulator $\mathcal{S}$ from effectively "rewinding" $\mathcal{A}$ in the execution of Π_1. This is achieved as follows. First, $\mathcal{A}$ aborts unless its output from Π_i is a valid signature on $F_k(\Pi_1(i-1)_1)$, where validity is with respect to vk_i. Second, in an ideal-model execution, $\mathcal{S}$ can obtain at most one signature that is valid with respect to the verification key vk_i. (This is because $\mathcal{S}$ does not know the signing key sk_i. Therefore, unless $\mathcal{S}$ can forge signatures, it can only obtain a signature by querying the trusted third party. However, the trusted party can only be queried once in each execution, and so only provides a single signature with key sk_i.) Third, by the collision resistance of pseudorandom functions,[9] $\mathcal{S}$ cannot generate a message $\Pi_1(i-1)'_1$ such that $F_k(\Pi_1(i-1)'_1) = F_k(\Pi_1(i-1)_1)$. Putting this together, we have that there is at most one $\Pi_1(i-1)_1$ message seen by $\mathcal{A}$ for which it does not abort. That is, $\mathcal{S}$ is unable to see $\mathcal{A}$'s response on different $\Pi_1(i-1)_1$ messages, or in other words, $\mathcal{S}$ is unable to rewind $\mathcal{A}$ in Π_1. We note that this heavily utilizes the first two features of the functionality f as described above.

The above explains why $\mathcal{S}$ cannot rewind $\mathcal{A}$ in execution Π_1. This, in itself, does not lead to any contradiction. However, recall that in order for $\mathcal{S}$ to simulate $\mathcal{A}$, it must send $\mathcal{A}$'s input to the trusted third party. This means that $\mathcal{S}$ must successfully extract $\mathcal{A}$'s input from all executions, and in particular from Π_1. We conclude that $\mathcal{S}$ can extract $\mathcal{A}$'s input in Π_1 without rewinding $\mathcal{A}$. Such an $\mathcal{S}$ can therefore be used by an adversary who controls P_1 and attacks P_2, in order to extract P_2's input (P_2 cannot be rewound, but $\mathcal{S}$ does not need to rewind it). Since P_2's input remains secret in f, this contradicts the security of the protocol. This last point utilizes the third feature of f described above; i.e., the fact that the signature is "blind".)

Ideal-model simulation of $\mathcal{A}$. By the assumption that Π is black-box secure, we have that there exists a black-box simulator $\mathcal{S}$ such that the result of an ideal execution with $\mathcal{S}$ is indistinguishable from the result of a real execution of $\mathcal{A}$ running $\Pi_1, \ldots, \Pi_m$. We now make some claims regarding the behavior of $\mathcal{S}$. Recall that $\mathcal{S}$ is given oracle access to the adversary $\mathcal{A}$. Thus, the oracle given to $\mathcal{S}$ is a next message function that receives a series of messages and computes $\mathcal{A}$'s response in the case that $\mathcal{A}$ received these messages in an execution. We note that if $\mathcal{A}$ would abort after receiving only a prefix of the messages in an oracle query, then its reply to the query is $\perp$. We now prove a

[9] We use pseudorandom functions and not collision resistant hash functions so that we can rely on the existence of one-way functions only.

special property of all oracle calls in which $\mathcal{A}$ does not abort.

Claim 3.1 *For every i, let Q_i be the set of all oracle queries sent by $\mathcal{S}$ to $\mathcal{A}$ during the ideal-model execution which include all messages from the Π_i execution and where $\mathcal{A}$ does not abort.*[10] *Then, except with negligible probability, the same message $\Pi_1(i-1)_1$ appears in every $q \in Q_i$.*

Proof: The proof of this claim is based on the security of the signature scheme and the collision resistance property of random (and thus pseudorandom) functions. First, we claim that except with negligible probability, $\mathcal{S}$ does not produce two oracle queries q and q' containing $\Pi_1(i-1)_1 \neq \Pi_1(i-1)'_1$, such that $F_k(\Pi_1(i-1)_1) = F_k(\Pi_1(i-1)'_1)$. Otherwise, we can construct a machine M that distinguishes F_k from a random function. Machine M works by emulating the entire experiment with $\mathcal{A}$ and $\mathcal{S}$, including choosing all inputs. The only difference is that instead of $\mathcal{A}$ computing $\alpha_i = F_k(\Pi_1(i-1)_1)$, machine M queries its oracle with $\Pi_1(i-1)_1$ and sets α_i to equal its oracle response. (Recall that M has oracle access to a function that is either truly random or equals $F_k(\cdot)$, for a random key k.) Now, on the one hand, if M has oracle access to $F_k(\cdot)$, then M's emulation for $\mathcal{S}$ is perfect. Therefore, with non-negligible probability M obtains from $\mathcal{S}$ two messages $\Pi_1(i-1)_1 \neq \Pi_1(i-1)'_1$ such that its oracle response is the same. On the other hand, if M has oracle access to a random function, then it will see such a collision with at most negligible probability. Therefore, with non-negligible probability, M distinguishes a pseudorandom function from a random one.

We now prove the claim under the assumption that such oracle queries q and q' are never produced by $\mathcal{S}$. Intuitively, in this case, the claim follows from the security of the signature scheme. That is, $\mathcal{S}$ is allowed to query the trusted party for the i^{th} execution only once, and this query provides $\mathcal{S}$ with a signature such that if the signature is on the message x where $x = F_k(\Pi_1(i-1)_1)$, then $\mathcal{A}$ does not abort. Now, if there are two oracle queries with different $\Pi_1(i-1)_1$ messages (and thus with different $F_k(\Pi_1(i-1)_1)$ values) such that $\mathcal{A}$ does not abort in either of them, then it must be that $\mathcal{S}$ provided $\mathcal{A}$ with valid signatures on both $F_k(\Pi_1(i-1)_1)$ values. Since only one signature could have been obtained from the trusted party, $\mathcal{S}$ must have generated a forgery.

More formally, we construct a forging algorithm M that is given vk and simulates $\mathcal{A}$ and the trusted party for $\mathcal{S}$. That is, M chooses the inputs for both parties in every execution Π_j for $j \neq i$. This means that M knows the signing-key sk_j for all of these executions. In contrast, the verification-key vk_i of the i^{th} execution is set to be the verification-key vk received by M. The forger M then perfectly emulates an ideal-model execution for $\mathcal{S}$. This involves emulating $\mathcal{A}$ and the trusted party. Note that the trusted party can be emulated with no problem in all executions except for Π_i (because M knows all the inputs, including the signing keys). In contrast, in order to emulate the trusted party for the i^{th} execution, M needs to be able to sign with the key that is associated with vk. M does this by querying its signing oracle. Thus, the emulation is perfect.

Now, assume that with non-negligible probability, there exist two queries $q, q' \in Q_i$ with different messages $\Pi_1(i-1)_1$ and $\Pi_1(i-1)'_1$, respectively. Since $\mathcal{A}$ does not abort, it must have received valid signatures for both $F_k(\Pi_1(i-1)_1)$ and $F_k(\Pi_1(i-1)'_1)$ under the verification key $vk_i = vk$. (Recall that in this case, $F_k(\Pi_1(i-1)_1) \neq F_k(\Pi_1(i-1)'_1)$.) Now, when $\mathcal{A}$ halts and its view includes the prefix q, its output includes a signature on $F_k(\Pi_1(i-1)_1)$ (because $\mathcal{A}$ outputs the output it received in all executions before it possibly aborts). Likewise, when $\mathcal{A}$ halts and its view includes the prefix q', its output includes a signature on $F_k(\Pi_1(i-1)'_1)$. We conclude that by continuing the emulation

[10] That is, we consider all of the oracle queries made by $\mathcal{S}$ to $\mathcal{A}$ throughout the simulation and take the subset of those queries which include all the messages of Π_i. By the scheduling described in Figure 1, such a query defines the i^{th} message of Π_1 that is sent by $\mathcal{A}$ (i.e., $\Pi_1(i)_2$).

with both these prefixes, M can obtain $(F_k(\Pi_1(i-1)_1), \sigma)$ and $(F_k(\Pi_1(i-1)'_1), \sigma')$ that constitute two valid message/signature pairs for the key vk. However, during the ideal-model execution, $\mathcal{S}$ can only query the trusted party once for the i^{th} execution. Therefore, M queried its signing oracle with at most one of these messages. The other pair thus constitutes a valid forgery and M succeeds with non-negligible probability, in contradiction to the security of the signature scheme. ■

Next, we claim that $\mathcal{S}$ must obtain the input z that is used by $\mathcal{A}$ in the execution of Π_1. That is,

Claim 3.2 *At some point in the ideal execution, except with negligible probability, $\mathcal{S}$ must send the trusted party the pair $(*, z)$ where z is $\mathcal{A}$'s auxiliary input and $*$ denotes any string.*

Proof: In a real execution of $\Pi_1, \ldots, \Pi_m$, the output of $\mathcal{A}$ from Π_1 equals σ_1 where $\mathsf{Verify}_{vk_1}(z, \sigma_1) = 1$. This is because $\mathcal{A}$ plays the honest party's strategy in this execution with input (vk_1, z), and thus the execution of Π_1 is exactly like an execution between two honest parties. (It is easy to show that in an execution of a secure protocol with two honest parties, except with negligible probability, the output of the parties must be according to the functionality definition.) Therefore, it follows that in an ideal execution with $\mathcal{S}$, except with negligible probability, the output of $\mathcal{A}$ must also be a valid signature σ_1 for z. This holds because, otherwise, a distinguisher D who receives all the parties' inputs and outputs, including $\mathcal{A}$'s auxiliary input z, could easily distinguish the real and ideal executions.

Now, in an ideal execution, $\mathcal{S}$ is given no information about sk_1 and z. Thus, if $\mathcal{S}$ queries the trusted party with $(*, z)$, it will obtain the necessary signature σ_1 on z. Otherwise, it must forge a signature on z, so that verification with the key vk_1 succeeds. Formally, if $\mathcal{S}$ can obtain a correct σ_1 without querying the trusted party with $(*, z)$, then we can construct a forging algorithm M who breaks the one-time signature scheme. The actual reduction is essentially the same as in the proof of Claim 3.1 and the details are therefore omitted. ■

Essentially, Claim 3.1 tells us that for every i, there is only a single $\Pi_1(i-1)_1$ message sent by $\mathcal{S}$ to $\mathcal{A}$ for which $\mathcal{A}$ does not abort. Thus, any "rewinding" of $\mathcal{A}$ is of no help (i.e., in any rewinding, $\mathcal{S}$ will either have to replay the same messages as before, or $\mathcal{A}$ will just abort). Furthermore, Claim 3.2 tells us that $\mathcal{S}$ succeeds in "extracting" the input z used by $\mathcal{A}$ in the execution of Π_1. These claims are central to our construction of an adversary $\mathcal{A}'$ who corrupts P_1 and uses $\mathcal{S}$ to "attack" P_2. (Notice that we are switching the identity of the corrupted party here.) The adversary $\mathcal{A}'$ will internally invoke $\mathcal{S}$ and forward all messages from Π_1 between $\mathcal{S}$ and the honest P_2 (instead of between $\mathcal{S}$ and $\mathcal{A}$). Now, since $\mathcal{S}$ has only black-box access to $\mathcal{A}$ and cannot rewind $\mathcal{A}$, its interaction with $\mathcal{A}$ is the same as its interaction with P_2. Recall further that $\mathcal{A}$ actually plays the role of the honest P_2 in Π_1, and so it makes no difference to $\mathcal{S}$ whether it is running an ideal execution with $\mathcal{A}$ or a real execution with P_2. This implies that if $\mathcal{S}$ can extract $\mathcal{A}$'s input in the ideal model execution, then $\mathcal{A}'$ (internally running $\mathcal{S}$) can extract P_2's input in a real execution. However, this will result in a contradiction because P_2's input should not be revealed in a secure protocol execution (by the definition of the functionality f).

The adversary $\mathcal{A}'$: Let $\mathcal{A}'$ be an adversary for a single execution of Π who corrupts P_1 and interacts with P_2. In this execution, P_1's input is a signing key sk and P_2's input is the associated verification key vk and a uniformly chosen $\alpha \in_R \{0,1\}^n$. $\mathcal{A}'$ *internally* incorporates $\mathcal{S}$ and emulates the concurrent executions of $\Pi_1, \ldots, \Pi_m$ while *externally* interacting in a single execution of Π with P_2. A technicality that arises with such a strategy is that $\mathcal{A}'$ is a strict polynomial-time machine, whereas $\mathcal{S}$ is allowed to run in expected polynomial-time. Therefore, let $q(n)$ be the polynomial that bounds the expected running time of $\mathcal{S}$. Then, if in the emulation, $\mathcal{S}$ exceeds $2q(n)$ steps, $\mathcal{A}'$ truncates the execution and outputs $\mathsf{time\text{-}out}$. This ensures that $\mathcal{A}'$

runs in strict polynomial-time. Furthermore, by Markov's inequality, $\mathcal{A}'$ outputs time-out with probability at most 1/2.

Notice that $\mathcal{S}$ interacts with a trusted third party and with $\mathcal{A}$; therefore, $\mathcal{A}'$'s emulation involves emulating both of these entities. $\mathcal{S}$ also expects to receive the input vector $(vk_1, \alpha_1, \ldots, vk_{p(n)}, \alpha_{p(n)})$ that $\mathcal{A}$ receives, and so $\mathcal{A}'$ must provide this in the emulation as well. (Note that $\mathcal{S}$ does *not* receive $\mathcal{A}$'s auxiliary input z.) Below, we will refer to P_2 as the *external* P_2 in order to differentiate between real interaction with P_2 and internal emulation.

The executions $\Pi_2, \ldots, \Pi_m$ are all internally emulated by $\mathcal{A}'$. That is, $\mathcal{A}'$ selects all the inputs (for both parties) and emulates both $\mathcal{A}$ and the trusted party for $\mathcal{S}$. This emulation can be carried out perfectly because $\mathcal{A}'$ knows all inputs, including the signing keys. However, the emulation of Π_1 is carried out differently. That is, $\mathcal{A}'$ sets $\mathcal{S}$'s input in Π_1 to be (vk, α') where vk is the verification key associated with sk and $\alpha' \in_R \{0,1\}^n$. (Recall that sk is $\mathcal{A}'$'s input in the single execution with the external P_2 and that, by assumption, sk explicitly contains a description of vk. Notice also, that α' has no correlation with the external P_2's input α.) Having set the input, we now describe how $\mathcal{A}'$ emulates $\mathcal{S}$'s interaction with $\mathcal{A}$ in Π_1. This emulation works by having the external P_2 play the role of $\mathcal{A}$. That is, let q be an oracle query from $\mathcal{S}$ to $\mathcal{A}$, such that $\mathcal{A}$'s response to q is the i^{th} message of execution Π_1. Then, if $\mathcal{A}$ would abort upon receiving q or any prefix of q, adversary $\mathcal{A}'$ emulates $\mathcal{A}$ aborting. (Note that $\mathcal{A}'$ can know if $\mathcal{A}$ would abort because this depends only on $\mathcal{A}$'s output from executions $\Pi_2, \ldots, \Pi_m$, which are all internally emulated by $\mathcal{A}'$.) Otherwise, q is such that $\mathcal{A}$ does not abort, but rather replies with the i^{th} message of Π_1 (i.e., $\Pi_1(i)_2$). Now, if the external P_2 has already sent the i^{th} message of Π, then $\mathcal{A}'$ replays this same message to $\mathcal{S}$ as $\mathcal{A}$'s reply. Otherwise, $\mathcal{A}'$ extracts the $\Pi_1(i-1)_1$ message from q and sends it to the external P_2, who replies with $\Pi_1(i)_2$. $\mathcal{A}'$ records this message and passes it to $\mathcal{S}$ as $\mathcal{A}$'s reply from the query q. This describes how the role of $\mathcal{A}$ in Π_1 is emulated. (The above description implicitly assumes that, without loss of generality, $\mathcal{S}$ queries $\mathcal{A}$ with all prefixes of q before querying q.) In order to complete the emulation, $\mathcal{A}'$ also needs to emulate the trusted party in Π_1. However, this is straightforward because $\mathcal{A}'$ knows the signing key sk (it is its input), and can therefore provide the output as generated by the trusted party. At the conclusion, $\mathcal{A}'$ outputs its view, including all the messages sent by $\mathcal{S}$ to the trusted third party. This completes the description of $\mathcal{A}'$.

Deriving a contradiction. We first claim that $\mathcal{S}$'s view in the emulation by $\mathcal{A}'$ is statistically close to its view in an ideal execution with oracle-access to $\mathcal{A}$. The emulation of $\mathcal{A}$ and the trusted third party in executions $\Pi_2, \ldots, \Pi_m$ is clearly identical. We now demonstrate statistical closeness for execution Π_1. First, note that the input distribution for Π_1 is the same. That is, $\mathcal{S}$ is given a pair (vk, α') for P_2's input, and the external P_2 really uses the pair (vk, α). (Recall that α and α' are independent uniformly distributed n-bit strings.) This is exactly the same as in an ideal execution where $\mathcal{S}$'s input in Π_1 equals (vk, α); yet $\mathcal{A}$ replaces α with z and really uses the pair (vk, z) for input. (Recall that α and z are also independent and uniformly distributed n-bit strings.) Second, we claim that the distribution over the messages sent by P_2 is statistically close to the distribution over the non-abort messages answered by $\mathcal{A}$. This follows from the fact that both $\mathcal{A}$ and P_2 follow the instructions of Π and from the following observation. Denote by Q the set of all oracle queries q which result in $\mathcal{A}'$ sending a message to the external P_2. Then, except with negligible probability, the same $\Pi_1(i)_1$ messages appear in every $q \in Q$, for every i. This follows directly from Claim 3.1; i.e., otherwise, with non-negligible probability, there exists a j for which Q_j contains two different $\Pi_1(j-1)_1$ messages. Thus, $\mathcal{A}$'s non-abort replies are consistent with an honest party who receives a

single series of messages $\Pi_1(1)_1, \ldots, \Pi_1(m)_1$. This also implies that the messages replayed from P_2 to $\mathcal{S}$ are the same as would be sent by $\mathcal{A}$ to $\mathcal{S}$, except with negligible probability. Third, $\mathcal{A}'$ sends $\perp$ to $\mathcal{S}$ in the emulation if and only if $\mathcal{A}$ would send $\perp$ to $\mathcal{S}$ in an ideal execution. Finally, note that since $\mathcal{A}'$ has the signing key sk, it plays the trusted party's role perfectly in the emulation. Combining the above, we have that $\mathcal{S}$'s view in the emulation by $\mathcal{A}'$ is statistically close to its view in an ideal execution. That is, all abort and non-abort oracle replies received by $\mathcal{S}$ in the emulation by $\mathcal{A}'$ are statistically close to what $\mathcal{S}$ would see in an ideal execution with $\mathcal{A}$. Furthermore, the output from the emulated trusted party that $\mathcal{S}$ receives from $\mathcal{A}'$ is identically distributed to the output that $\mathcal{S}$ would receive from the real trusted party in an ideal execution. In conclusion, all messages seen by $\mathcal{S}$ in the two scenarios are statistically close.

Now, by Claim 3.2, except with negligible probability, $\mathcal{S}$ must send $(*, z)$ to the trusted third party at some stage of the ideal execution, where z is the auxiliary input of $\mathcal{A}$ that it uses as input in Π_1. Therefore, with probability at most negligibly less than $1/2$, $\mathcal{S}$ must send $(*, \alpha)$ in the above emulation by $\mathcal{A}'$, where α is the input used by P_2 in Π. In such a case, α is included in $\mathcal{A}'$'s output. (Recall that $\mathcal{A}'$ truncates $\mathcal{S}$'s execution after $2q(n)$ steps. Therefore, with probability $1/2$, $\mathcal{S}$ may not terminate and so may not output $(*, z)$. This is the reason that $\mathcal{A}'$ obtains α with probability only $1/2$.) In summary, in a real execution of $\mathcal{A}'$ with P_2, adversary $\mathcal{A}'$ obtains P_2's input with probability almost $1/2$. Intuitively, this yields a contradiction because P_2's input is not revealed by the functionality definition. Formally, consider the ideal-model execution for this scenario; let $\mathcal{S}'$ be the ideal-model adversary that is guaranteed to exist for $\mathcal{A}'$ in Π. By the security of the protocol, it must be that $\mathcal{S}'$ outputs P_2's input α with probability at most negligibly less than $1/2$. (Otherwise, the real and ideal executions can be easily distinguished.) However, in such an ideal execution, $\mathcal{S}'$ is given no information on P_2's input α. Since α is uniformly distributed in $\{0,1\}^n$ it follows that $\mathcal{S}'$ can output this value with at most negligible probability. We therefore conclude that such a protocol Π does not exist. ■

Notice that Theorem 2 holds if the simulator for the case that P_2 is corrupted is black-box, irrespective of whether or not the simulator for a corrupt P_1 is also black-box (of course, the roles of P_1 and P_2 can be reversed). We remark that for our upper bound (Theorem 1.3), one corruption case is proven using black-box simulation and the other is proven using non black-box simulation. Therefore, Theorem 2 states that any protocol using such a strategy must have round complexity greater than m.

Impossibility of (black-box) unbounded concurrent composition. The following important corollary demonstrates a severe limitation of black-box simulation in the setting of concurrent secure two-party computation:

Corollary 3 *There exists a probabilistic polynomial-time functionality f for which there does not exist a protocol that securely computes f under* unbounded *concurrent composition and is proven using* black-box simulation.

We stress that this does not rule out the possibility of obtaining concurrent protocols for specific (non-trivial) functionalities; rather, our result holds for the specific "blind signature" functionality and thus also for general protocols.

Non-uniform versus uniform adversaries. Our proof of the lower bound uses the non-uniformity of the adversary $\mathcal{A}$ in an essential way. That is, we require that $\mathcal{A}$ use its auxiliary input z in the execution of Π_1. This is important because, on the one hand, the simulator $\mathcal{S}$ must

not have access to the input used by $\mathcal{A}$. On the other hand, the distinguisher must know which input is used by $\mathcal{A}$ in order to check that the signature generated by $\mathcal{S}$ is on the "correct" input. We solve this by using $\mathcal{A}$'s auxiliary input which is not given to $\mathcal{S}$ (but *is* given to the distinguisher). We remark that we do not know whether or not Theorem 2 holds also for uniform adversaries.

3.2 Impossibility For Concurrent Oblivious Transfer

Although Theorem 2 and Corollary 3 suffice for ruling out general protocols, we cannot infer a lower bound from them to any specific functionality other than that of blind signatures. Nevertheless, we do show that, as a corollary to the above proof, concurrent secure computation of the oblivious transfer functionality *can* be ruled out. Recall that 1-out-of-2 oblivious transfer is defined by: $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$, where $x_0, x_1 \in \{0,1\}^n$ and $\sigma \in \{0,1\}$ [15].

Corollary 4 *There does not exist a protocol that securely computes the* 1-out-of-2 oblivious transfer *functionality under unbounded concurrent composition and is proven using black-box simulation.*

Proof Sketch: We show that a protocol for 1-out-of-2 oblivious transfer that remains secure under unbounded concurrent composition can be used to obtain a protocol for the blind signature functionality that remains secure under unbounded concurrent composition. Since we have already proven an impossibility result for the blind signature functionality, this implies an impossibility result for the oblivious transfer functionality as well.

We now describe the protocol for blind signatures that uses the 1-out-of-2 oblivious transfer functionality. We remark that since the concurrent setting that we consider is where only a single protocol is run many times concurrently, our protocol for blind signatures must use invocations of the oblivious transfer functionality and *nothing else*. Our protocol uses the specific construction of [31] for a one-time signature schemes. The signature scheme of [31] is defined as follows. Let f be a one-way function. Then, the signing-key equals $2n$ random points $x_1^0, x_1^1 \ldots, x_n^0, x_n^1$, and the verification key equals $y_1^0, y_1^1 \ldots, y_n^0, y_n^1$, where for every i, $y_i^0 = f(x_i^0)$ and $y_i^1 = f(x_i^1)$. Now, a signature on the message $w = w_1 \cdots w_n \in \{0,1\}^n$ equals $x_1^{w_1}, \ldots, x_n^{w_n}$. The verification of a signature is carried by simply checking that for every i, $f(x_i^{w_i}) = y_i^{w_i}$. We are now ready to present the protocol:

Protocol 1 (blind signatures from oblivious transfer):

- Inputs: *Party* P_1 *(the signer) has* $(x_1^0, x_1^1 \ldots, x_n^0, x_n^1)$*, and* P_2 *has* $(y_1^0, y_1^1 \ldots, y_n^0, y_n^1)$ *and* $w \in \{0,1\}^n$.
- The Protocol:
  1. P_1 *and* P_2 *run* n *copies of a protocol for 1-out-of-2 oblivious transfer that remains secure under concurrent composition.*
  2. *In the* i^{th} *execution,* P_1 *inputs* (x_i^0, x_i^1) *and* P_2 *inputs* w_i.
  3. *At the conclusion,* P_2 *outputs* $x_1^{w_1}, \ldots, x_n^{w_n}$.

The proof of security of the protocol is straightforward. That is, in the case that P_1 is corrupted, the ideal-model simulator/adversary obtains all the x_i^b's from $\mathcal{A}$. This is because $\mathcal{A}$ must send each x_i^b to the oblivious transfer functionality. These strings constitute the entire signing key and thus the simulator sends them to the trusted party. Likewise, in the case that P_2 is corrupted, the w_i's sent to each copy of the oblivious transfer protocol constitute the entire input w. Therefore, the ideal-model simulator can send the input to the trusted party and obtain the required

signature. The key point is that since the oblivious transfer functionality remains secure under unbounded concurrent composition, the same is true of the blind signature functionality. However, this contradicts Corollary 3.

We note that the exact lower bound on the number of rounds of oblivious transfer derived above is as follows: any protocol that securely computes the oblivious transfer functionality under m-bounded concurrent composition must have more than m/n rounds. This loss of a factor of n is due to the fact that n copies of the oblivious transfer functionality are needed for every blind signature execution. ■

Remark. Notice that Corollary 4 seems to contradict the existence of a protocol for concurrent oblivious transfer as shown in [18]. However, the model of concurrency of [18] is different to ours in an essential way. Specifically, they assume that the inputs of the parties are chosen independently in every execution. Formally, they define the distribution over the inputs as follows: The inputs of P_1 and P_2 are chosen according to $2p(n)$ efficiently samplable distributions $X_1, \ldots, X_{p(n)}$ and $Y_1, \ldots, Y_{p(n)}$, respectively. Furthermore, the random coins used in each X_i and Y_j are *independent*. This means that P_1 and P_2's inputs are not correlated, and that each party's inputs in different executions are also not correlated.

In contrast, our model follows the more standard definition where quantification is over all inputs (and, in particular, over correlated inputs). Specifically, our definition implies security even in the case that a *single* distribution $\overline{XY}$ outputs all $2p(n)$ of the party's inputs (using a single random tape). Furthermore, we use this correlation in an essential way in our lower bound.

3.3 Extensions of the Lower Bounds

We now show extensions of our lower bounds to more relaxed definitions of security.

Restricting the input correlations. As described above, the current formulation of the lower bound considers a case where all inputs can be correlated. However, the bound can also be shown to hold when some independence of inputs exists. We consider a number of possible models:

1. First consider a model where the inputs of P_1 and P_2 may be correlated in any given execution, but the inputs between different executions are independent. Formally, in this model the inputs are chosen according to a series of (known) distributions $XY_1, \ldots, XY_{p(n)}$, where for each i, $XY_i(r_i)$ outputs a pair of inputs (x_i, y_i) for the parties, and for every $i \neq j$, the random coins used by XY_i and XY_j are independent. We claim that the lower bound holds also for this model. In fact, the proof of Theorem 2 holds without any modification. This follows from the fact that the input distribution considered in the proof is such that the inputs in each execution are chosen independently of each other. Specifically, in each execution, we choose a pair of signature keys $(vk, sk) \leftarrow \mathsf{Gen}(1^n)$ and a random string $\alpha \in_R \{0,1\}^n$. Then, P_1's input is set to sk, and P_2's input is set to (vk, α). Furthermore, in every execution, new signing keys are used. Therefore, the random coins used in generating the inputs are independent in every execution.

2. In the above model, the parties have correlated inputs, but the auxiliary input of the adversary is independent. However, one can also consider a model where the parties' inputs are independent of each other, but the adversary's auxiliary input may be correlated. That is, define the input distributions as $X_1, Y_1, Z_1, \ldots, X_{p(n)}, Y_{p(n)}, Z_{p(n)}$, where the random coins used in each X_i and Y_j are independent (including when $i = j$). However, Z_i may be correlated to

X_i and Y_i (formally, give Z_i the coins r_i^x and r_i^y used by X_i and Y_i, respectively). As above, the proof of Theorem 2 also works here. The only difference is that $\mathcal{A}$'s auxiliary input is defined to be $(z, vk_1, \ldots, vk_m)$, where $z \in_R \{0,1\}^n$ and vk_i is the verification key associated with sk_i that P_1 receives for input in the i^{th} execution. In such a case, the input of P_2 does *not* include vk_i (since this would constitute correlated input); rather, P_2 receives a random α_i only. The rest of the proof remains the same.

In contrast to the above models where Theorem 2 holds, consider a further relaxation where the inputs of the parties are chosen independently of each other and independently of the inputs in other executions. This is the model considered by [18]. Since they demonstrated the feasibility of oblivious transfer in the model, we know that Theorem 2 does *not* hold here. We remark that this model is rather restricted as it does not allow any correlation of inputs between the parties, or any correlated auxiliary knowledge given to the adversary.

Quasi-polynomial time simulation. Recently, it was suggested that quasi-polynomial time simulation can be used to solve the difficulties arising in the concurrent composition of protocols [34]. However, assuming sub-exponential hardness, our lower bounds also hold when the simulator is allowed to run in quasi-polynomial time; i.e., in time $n^{\text{polylog}(n)}$. Specifically, the computational primitives that we use in our proof are pseudorandom functions and one-time signature schemes. If the security of these primitives holds even in the face of quasi-polynomial time adversaries, then Theorem 2 and Corollaries 3 and 4 hold even when quasi-polynomial time simulation is allowed. We note that such security can be obtained assuming the existence of one-way functions that are hard to invert in time 2^{n^ϵ}, for some constant ϵ.

4 Tools for our Protocol – Zero-Knowledge

In this section, we develop the basic tools needed for proving Theorem 1.3 (our positive result). We use standard definitions for zero-knowledge; see [20].

4.1 The Zero-Knowledge Proof of Knowledge of [4]

The zero-knowledge arguments of knowledge of [4] have the interesting property that *both* simulation and extraction take place by simulating zero-knowledge proofs (that are used as subprotocols). That is, the argument system is such that both the prover and verifier prove zero-knowledge proofs (of membership) during the protocol. Then, the simulator strategy essentially just involves the simulation of the subproof given by the prover to the verifier. Likewise, the extraction strategy essentially just involves the simulation of the subproof given by the verifier to the prover. This is helpful because we can focus on the one issue of simulating zero-knowledge within our setting, and we obtain both zero-knowledge simulation *and* knowledge extraction.

Commit-with-extract commitment schemes. A central tool in the construction of the zero-knowledge arguments of knowledge of [4] is a perfectly-binding commitment scheme with the following *extraction* property: for every committer, there exists a commitment *extractor* who is able to extract the value being committed to by the committer. We first describe how such a scheme helps in obtaining zero-knowledge arguments of knowledge:

Zero-knowledge arguments of knowledge using commit-with-extract. Given a commit-with-extract commitment scheme, a zero-knowledge argument of knowledge can be constructed as follows. The basic idea is that the prover commits to a witness using the commit-with-extract scheme, and then proves that it committed to a valid witness using a zero-knowledge proof of membership. Intuitively, a knowledge extractor for the argument of knowledge simply uses the extractor of the commit-with-extract scheme. By the soundness of the proof of membership in the second stage of the argument of knowledge, we are guaranteed that the value obtained will be a correct witness with probability only negligibly less than the probability that the verifier accepts the proof. A simulator for this protocol is also easily obtained: just commit to garbage in the commit-with-extract scheme and then simulate the proof of membership in the second stage. See [4] for more details. The protocol description is as follows:

Protocol 2 (zero-knowledge argument of knowledge for $R \in \mathcal{NP}$):

- **Common Input:** x
- **Auxiliary input to prover:** *w such that $(x, w) \in R$.*
- **Part 1:** *P and V run a commit-with-extract protocol in which P commits to the witness w.*
- **Part 2:** *P proves to V that it committed to a valid witness w in the previous step, using a zero-knowledge proof (or argument) of membership.*[11]

The commit-with-extract scheme of [4]. The scheme is based on the following non-interactive commitment scheme that uses one-way permutations: Let f be a one-way permutation and let b be a hard-core predicate of f. Then, in order to commit to a bit σ, the committer chooses $r \in_R \{0,1\}^n$, lets $y = f(r)$ and sends $\langle y, b(r) \oplus \sigma \rangle$ to the receiver. Loosely speaking, the commit-with-extract scheme is defined in the same way except that the value $y = f(r)$ is chosen jointly by the committer and the receiver using a coin-tossing protocol. Since y is chosen uniformly, the hiding property remains as in the original scheme. Likewise, because f is a permutation, y defines a unique value $b(f^{-1}(y))$ and thus the scheme remains perfectly binding. The novelty of the scheme is that the extractor can *bias* the outcome of the coin-tossing protocol so that it knows the preimage $f^{-1}(y)$ of y. In this case, the extractor can easily obtain the commitment value σ, as desired. We now describe the protocol:

Protocol 3 (commit-with-extract commitment scheme):

- **Input:** *The committer has a bit σ to be committed to.*
- **Commit phase:**
  1. The committer chooses an enhanced trapdoor permutation:[12]

     The committer chooses an enhanced trapdoor permutation f along with its trapdoor t and sends f to the receiver.[13]

[11] Formally, let trans be the transcript of the commit-with-extract execution of part 1, and let d be the decommitment message that P would use to decommit. Then, P proves the NP-statement that there exists a value d such that (trans, d) defines the value w, and $(x, w) \in R$.

[12] See [21, Appendix C] for the definition of *enhanced* trapdoor permutations. For the sake of understanding the protocol here, it suffices to think of the case that the domain of the permutation is $\{0,1\}^n$.

[13] For the sake of simplicity, we assume that the protocol uses *certified* [17] enhanced trapdoor permutations (for which the receiver can efficiently verify that f is indeed a permutation). However, actually any enhanced trapdoor permutation can be used; see [4].

2. The parties run a coin-tossing protocol:
   (a) *The receiver chooses a random string* $r_1 \in_R \{0,1\}^n$ *and sends* $c = \mathsf{Commit}(r_1) = C_n(r_1; s)$ *to the committer* (*using any perfectly-binding commitment scheme* C_n *and a random string* s).
   (b) *The committer chooses a random string* $r_2 \in_R \{0,1\}^n$ *and sends* r_2 *to the receiver.*
   (c) *The receiver sends* r_1 *to the committer* (*without decommitting*).
   (d) *The receiver proves that* r_1 *is the value that it indeed committed to, using a zero-knowledge argument system. Formally, the receiver proves that there exists a string* s *such that* $c = C_n(r_1; s)$.
   (e) *The output of the coin-tossing phase is* $r_1 \oplus r_2$.
3. The actual commitment:
   The committer computes $r = f^{-1}(r_1 \oplus r_2)$ *and sends the value* $v = b(r) \oplus \sigma$ *to the receiver.*

- **Reveal phase:**
  1. *The committer sends the receiver the string* r.
  2. *The receiver checks that* $f(r) = r_1 \oplus r_2$. *If this is the case, then it computes* $b(r) \oplus v$ *obtaining* σ. *Otherwise, it outputs* $\perp$.

First note that if the coin-tossing protocol ensures that $r_1 \oplus r_2$ is pseudorandom, then the hiding property of the commitment scheme is preserved. This is because the receiver cannot predict $b(r)$ from a (pseudo-)randomly chosen $f(r)$ with probability greater than $1/2$. We remark that as long as the proof given by the receiver is *sound*, the security of the coin-tossing protocol holds.

We conclude by briefly describing the commitment extractor CK for the protocol. CK works by biasing the outcome $r_1 \oplus r_2$ of the coin-tossing protocol, so that it knows the preimage under f. More specifically, CK chooses a random string r, computes $f(r)$ and then makes the output $r_1 \oplus r_2$ equal $f(r)$. This is clearly not possible for a real receiver to do (as the coin-tossing protocol ensures that $f(r)$ is pseudorandom). However, CK can run the simulator for the proof that B provides in step 2d of the protocol. Thus, CK sends a commitment to garbage in step 2a of the protocol and waits to receive the string r_2 from the committer. Upon receiving r_2, extractor CK computes $r_1 = f(r) \oplus r_2$ and sends r_1 to the committer (where $f(r)$ is obtained by choosing a random r and applying f to r). Now, with overwhelming probability, this is not the string that CK committed to earlier. Thus CK cannot prove the proof of step 2d. Instead, it runs the simulator for it (and by the hiding property of commitments, this looks indistinguishable from a real execution). The key point is that at the conclusion of the protocol, the committer defines $f(r) = r_1 \oplus r_2$ and sends $v = b(r) \oplus \sigma$. However, CK knows the string r and can therefore obtain σ by computing $v \oplus b(r)$. Notice that the extraction strategy relies only on the ability to simulate zero-knowledge proofs.

We remark that the extractor for the zero-knowledge protocol can be made to extract a valid witness with probability that is negligibly close to the probability that the honest verifier accepts the proof. This extractor works by running the extractor of the commit-with-extract scheme (that always extracts the value committed to). The extractor then verifies the proof of membership of part 2, playing the honest verifier. If it accepts the proof, the extractor outputs the value obtained. Otherwise, it outputs a special symbol $\perp$ that is not a valid witness for any statement.

4.2 Black-Box Bounded Concurrent ZK

In this section we describe the concurrent zero-knowledge protocol of Prabhakaran et al. [35] with a small modification. Whereas the [35] protocol has $\widetilde{O}(\log n)$ rounds, we extend the protocol to

have $O(m)$ rounds, where m is the maximum number of concurrent executions. Although the original [35] protocol is concurrent zero-knowledge for any (unbounded, yet polynomial) number of concurrent executions, their simulation strategy is problematic when the protocol is used as a module within another larger protocol (as will be the case here). We therefore modify the original protocol (only with respect to the number of rounds) and provide a *new* simulation strategy that also succeeds when the protocol is plugged into another (larger) protocol. We remark that our proof that the resulting protocol is m-bounded concurrent zero-knowledge is far simpler than all known proofs for the full concurrency case. (Of course this is facilitated by the fact that in the model of bounded concurrency we can make the protocol depend on the number of concurrent executions.) We now present the modified protocol that we denote PRS:

Protocol 4 (Protocol PRS – concurrent ZK proof of Hamiltonicity):

- **Common input:** *a graph G and a security parameter n.*
- **Preamble length:** k
- **Part 1 – the preamble:**
  1. *P sends V the first message of a two-round perfectly hiding commitment scheme.*
  2. *V chooses $q \in_R \{0,1\}^n$ and two series of $n \cdot k$ random strings $\{q^0_{i,j}\}$ and $\{q^1_{i,j}\}$ such that $q^0_{i,j} \oplus q^1_{i,j} = q$ for every i, j $(1 \leq i \leq n, 1 \leq j \leq k)$. Then, V sends P perfectly-hiding commitments to all of the above strings.*
  3. *Repeat k times (for $j = 1, \ldots, k$):*
     (a) P chooses $b_j = b_{1,j}, \ldots, b_{n,j} \in_R \{0,1\}^n$ and sends b_j to V.
     (b) V decommits to $q^{b_{1,j}}_{1,j}, \ldots, q^{b_{n,j}}_{n,j}$.
- **Part 2 – the proof:**
  1. *P sends the first message of the proof of Hamiltonicity of Blum* [6]*; see Appendix A.*
  2. *V decommits to q, $\{q^0_{i,j}\}$ and $\{q^1_{i,j}\}$ for every i, j.*
  3. *P verifies the validity of the decommitments, and that $q^0_{i,j} \oplus q^1_{i,j} = q$ for every i, j. If yes, P sends the second prover-message of the proof of Hamiltonicity, where the verifier query string is taken as q.*

Denote by PRS(k) the protocol where the number of iterations in the preamble equals k. Then, as we have mentioned, Prabhakaran et al. showed that PRS($\widetilde{O}(\log n)$) is unbounded concurrent zero-knowledge [35]. Nevertheless, we provide a simple proof to the following weaker claim:

Proposition 5 *Assuming the security of the described commitment schemes, Protocol* PRS($2m$) *is an m-bounded (black-box) concurrent zero-knowledge proof system for Hamiltonicity.*

Proof: In order to prove this proposition, we need to show completeness, soundness and m-bounded concurrent Zero-Knowledge. Completeness follows immediately from the completeness of the Blum's proof of Hamiltonicity in part 2 [6]. We proceed to prove soundness:

Soundness: By the soundness of the proof of Hamiltonicity of Blum, it suffices to show that before the verifier query string q is revealed in part 2, it is uniformly distributed (or statistically close to uniform) relative to P's view. However, this follows immediately from the fact that V commits using a perfectly hiding commitment scheme. Notice that during the preamble, for every

i, j at most one of $q^0_{i,j}$ and $q^1_{i,j}$ is revealed. Therefore, P obtains no information about q from these decommitments.

***m*-bounded concurrent zero-knowledge:** We begin by defining some notation and terminology. Denote by Π_k the k^{th} execution of the protocol and by Π^j_k the j^{th} iteration of the preamble in the k^{th} execution. We say that iteration Π^j_k opens when the prover P sends $b_j \in_R \{0,1\}^n$ and that it closes when the verifier V^* decommits to $q^{b_{1,j}}_{1,j}, \ldots, q^{b_{n,j}}_{n,j}$. An execution is said to have been satisfied by the simulator S if V aborts the execution (by say, refusing to decommit in some iteration), or if there exists an i and j such that S received decommitments to *both* $q^0_{i,j}$ and $q^1_{i,j}$ (recall that $q = q^0_{i,j} \oplus q^1_{i,j}$ and thus receiving both of these decommitments means that q is revealed). Notice that once an execution has been satisfied, the simulator can complete the simulation without any rewinding. In the case of an aborted execution, this is immediate. In the case that q is revealed, it is possible for S to generate a prover commitment that convinces the verifier, without knowing a Hamiltonian cycle; see Appendix A. (Actually, this holds only if V^* does not decommit later to a different q'. However, this follows from the computational binding of the commitment scheme.)

Loosely speaking, the simulator S works by playing the honest prover until the first iteration of an unsatisfied execution closes. Assume that this iteration is Π^j_k. Then, S rewinds to the point in execution k that iteration j opened and sends a new randomly chosen string b_j to V^*. Next, S continues forward (playing the honest prover) hoping that once again Π^j_k is the first iteration to close. If this occurs and V^* decommits, then with overwhelming probability execution k is satisfied. (Execution k may not be satisfied in the case that the new string b_j equals the old one. However, this happens with probability 2^{-n} only. In all other cases, there exists at least one i for which the new and old b_j's differ. For all such i's, S receives $q^0_{i,j}$ and $q^1_{i,j}$ and thus the execution is satisfied.) If Π^j_k is not the first iteration to close, then S repeats the process again, until it is. We remark that the expected number of such rewinds is inversely proportional to the probability that Π^j_k was the first potential iteration to close the first time. The simulation then continues in the same manner until all executions are satisfied. We now proceed to formally define the simulator:

THE SIMULATOR S: We describe the simulation strategy of S for the remaining set of unsatisfied executions (initially this set contains all executions). S's high-level strategy is to satisfy an execution, permanently fix the transcript until the point that the execution was satisfied (without any further rewinding behind this point), and then continue to satisfy another execution from that point on. Before continuing, we introduce some terminology. We call the point at which S fixes the transcript a fixed point.[14] Furthermore, the current fixed point is the last fixed point to be set by S. Now, S holds a list of unsatisfied executions and attempts to satisfy an execution from this list without rewinding behind its current fixed point. According to this strategy, only iterations that were opened after the current fixed point may be rewound. Thus, we call an iteration potential if it *opened* after the current fixed point. Now, S interacts with $V^*(x, z, r)$ in a black-box manner, playing the honest prover strategy and waiting for one of the following three events:

1. *S receives the verifier commitments $q, \{q^0_{i,j}\}, \{q^1_{i,j}\}$ in some execution:* When this happens, S fixes the transcript to this point (i.e., S re-marks the current fixed point to be the message containing the verifier commitments) and continues as above. (No execution has been satisfied; however, we never wish to rewind behind such messages.)

2. *V^* aborts an unsatisfied execution:* When this occurs, the execution is satisfied. Therefore, S sets a fixed point at the "abort" message, removes this execution from the set of unsatisfied

[14] Formally, "fixing the transcript" means fixing the prefix of all future oracle calls to V^*.

executions and continues as described above. We note that an abort message is just any detectably illegal message sent by V^*.

3. *A potential iteration closes:* Let this iteration be Π_k^j and denote the simulator-generated prover message of this iteration by b_j^{old}. Furthermore, let r be the random coins used by S in the simulation since the last fixed point was set. Now, S attempt to learn $q_{i,j}^0$ and $q_{i,j}^1$ for some i by rewinding V^* back to the current fixed point and playing the honest prover strategy again, using *new* random coins. (We stress that S does not rewind V^* back only to the opening of Π_k^j, but rather rewinds V^* all the way back to the current fixed point.[15]) Now, S's aim in these rewindings is to once again have Π_k^j be the first potential iteration to close. Therefore, if V^* sends verifier commitments, aborts an execution, or closes a different potential iteration *before* Π_k^j closes, then S rewinds back to the current fixed point and tries again (with new random coins). S continues until Π_k^j is the first iteration to close (without the other events occurring). Let b_j^{new} be the simulator-generated prover message of Π_k^j when this occurs. Then, if $b_j^{\text{new}} = b_j^{\text{old}}$, simulator S halts outputting a special failure symbol. Otherwise, there exists at least one i for which $b_{i,j}^{\text{new}} \neq b_{i,j}^{\text{old}}$. Since V^* decommitted according to both b_j^{new} and b_j^{old}, we have that S received both $q_{i,j}^0$ and $q_{i,j}^1$ (and thus can compute $q = q_{i,j}^0 \oplus q_{i,j}^1$). Therefore, the execution is satisfied.

   Having satisfied Π_k, simulator S rewinds V^* to the current fixed point one more time and replays the honest prover strategy using the random coins r that it used the first time that Π_k^j closed. Then, S fixes the transcript at the point that Π_k^j closes, removes Π_k from the set of unsatisfied executions and continues as above. (We remark that Π_k^j definitely closes because the same coins are used as before and thus the process is deterministic.)

S continues in this way until V^* concludes the interaction. However, there are two bad events that can happen during the described simulation:

1. *Part 2 of the proof of an* unsatisfied *execution is reached:* If this happens, then S is stuck because it does not know q and cannot prove the proof. Therefore, S halts outputting a special unsatisfied symbol.

2. *V^* decommits to a different $q' \neq q$ in part 2 of the proof:* That is, let q be the string obtained by S in some satisfied (non-aborted) execution and let $q' \neq q$ be the string decommitted to by V^* in part 2 of the proof of that execution, If this happens, then once again S cannot complete the proof (because it only knows how to answer the query string q and $q' \neq q$). Therefore, S halts and outputs failure.

This completes the description of S.

Proof of the simulation strategy: We begin by showing that the expected running-time of S is polynomial in n.

Claim 4.1 *The expected-time taken by the simulator S to set each fixed point is polynomial in n.*

Proof: At any stage of the simulation there are at most $3m$ possible events that can cause S to set a fixed point:

[15] This simplifies the analysis later and explains why fixed points are also set after verifier commitments are sent. Specifically, if the current fixed point happened to be before the verifier commitments of Π_k were sent, then after a rewinding, V^* may change the q and $q_{i,j}^b$ values, rendering the rewinding useless.

1. *V^* sends the verifier commitments in one of the executions:* The time it takes for S to deal with each of these (m possible) events is polynomial in the number of messages sent since the previous fixed point. Specifically, S plays the honest prover strategy in part 1 of some executions and uses the simulation strategy of part 2 of other executions (when given q). Since all of these messages can be computed in a fixed polynomial-time and there are at most a fixed polynomial number of messages (m times the number of rounds in PRS), this event contributes at most a fixed polynomial amount of work.

2. *One of the unsatisfied executions is aborted by V^*:* As in the previous case, S does a fixed polynomial amount of work here. Also, as above, the number of such possible events is at most m.

3. *A potential iteration in any of the (unsatisfied) executions is closed:* Unlike the previous cases, here S may do a lot of work. We first remark that there are only m different events that can occur here. This holds because each unsatisfied execution can have only one potential iteration between any two fixed points. (As soon as the first potential iteration closes, a new fixed point is set. Therefore, no execution can open two iterations between two fixed points.) Thus, the k^{th} event is the case that the potential iteration that closes belongs to execution Π_k. We proceed to show that the expected-time taken by S in the simulation for each of these events is polynomial in n. The sum of all these times is then a (coarse) upper bound on the expected time taken to set a fixed point due to such an event.

   Let p_k^j be the probability that Π_k^j is the potential iteration that closes before any other event causing a fixed point occurs. (As we have mentioned, this probability is non-zero for only one iteration j in execution k.) We remark that this probability is a function of V^*'s input tapes (x, z, r) and the transcript t that is fixed until this point and that the probability is taken over S's coins only. Now, the expected number of repetitions by S until it satisfies Π_k or outputs $\mathsf{failure}$ is exactly $1/p_k^j$. This is because S plays the honest prover strategy[16] for *all* repetitions, with the only difference being that new uniformly chosen coins are used in each repetition. In each repetition S does a polynomial amount of work. Therefore, the overall contribution to the expected-time is

$$p_k^j \cdot \frac{1}{p_k^j} \cdot \text{poly}(n) = \text{poly}(n)$$

The proof is completed by summing over the expected time for all of the $3m$ possible events. We thus have that the expected time taken by the simulator to set a fixed point is bounded by $O(m \cdot \text{poly}(n)) = \text{poly}(n)$. ■

Now, recall that in the simulation, S places at most $2m$ fixed points, where $m = \text{poly}(n)$. Therefore, the following claim is derived from Claim 4.1 and the linearity of expectations:

Claim 4.2 *The simulator S runs in expected-time that is polynomial in n.*

Next, we demonstrate that S outputs a distribution that is computationally indistinguishable from a real execution between P and V^*. We first bound the probability that S outputs a special $\mathsf{failure}$ or $\mathsf{unsatisfied}$ symbol.

[16] To be exact, S may deviate from the honest prover strategy if part 2 messages are sent during the rewinding. However, the important point to notice is that even in this case, S generates exactly the same distribution in the first attempt and in later rewindings (because it deviates in the same way also in the first attempt). The expected number of repetitions is therefore exactly $1/p_k^j$.

Claim 4.3 *The probability that S outputs* failure *is negligible.*

Proof: S outputs failure if for some j it happens that $b_j^{\text{new}} = b_j^{\text{old}}$ or if V^* decommits to some $q' \neq q$ in part 2 of the proof. Now, the probability that S outputs failure due to the first reason is negligible. This is because $b_j^{\text{new}} = b_j^{\text{old}}$ can happen only if the same random n-bit string is chosen in two rewinding attempts (when simulating the case that a potential iteration closes first). Since S runs in expected polynomial-time (as already proven), there exists a polynomial $q(\cdot)$ such that the probability that S exceeds $q(n)$ steps is at most $\mu(n)$, where $\mu(\cdot)$ is a negligible function. In particular, this means that except with negligible probability, the number of rewinding attempts overall is bound by $q(n)$. We therefore have that the probability that there exists a j for which $b_j^{\text{new}} = b_j^{\text{old}}$, is bounded by $q(n)/2^n + \mu(n)$, which is negligible.

Next, we claim that the probability that V^* decommits to $q' \neq q$ is negligible. This is shown by reducing this to the computational binding of the commitment scheme. The proof of this is straightforward and is therefore omitted. (An identical argument is shown in [23, Claim 3].) ■

Next, we show that S never outputs the special unsatisfied symbol.

Claim 4.4 *The probability that S outputs* unsatisfied *equals* 0.

Proof: S only outputs unsatisfied in the case that it reaches part 2 of the proof in an execution that it did not satisfy. We can assume that S does not output failure (because if it did, then it would not output unsatisfied). This implies that whenever S finds a potential iteration that closes, the execution to which this iteration belongs is satisfied. Thus, it suffices to show that every execution which is not aborted has a potential iteration which is the first to close after some fixed point.

This follows from the following argument. Assume that Π_k is not satisfied when it reaches part 2 of its proof. Now, there are $2m$ iterations in Π_k's preamble. Furthermore, as we have mentioned above, every unsatisfied execution can only open at most one iteration between every two fixed points. Therefore, S must have placed $2m$ fixed points before part 2 of Π_k is reached. However, S only places fixed points when verifier commitments are sent (at most m) and when executions are satisfied (at most m, including Π_k). Thus, Π_k must have been satisfied. ■

Having established that S outputs a special symbol with at most negligible probability, we are ready to prove that the simulation is successful. That is,

Claim 4.5 *Let L be the language of Hamiltonian graphs. Then, for every polynomial-time verifier V^*, it holds that*

$$\left\{\langle P, V^*(z,r)\rangle(G)\right\}_{G\in L, z,r\in\{0,1\}^*} \stackrel{\text{c}}{\equiv} \left\{S^{V^*(G,z,r)}(G)\right\}_{G\in L, z,r\in\{0,1\}^*}$$

Proof: We assume that S does not output failure or unsatisfied. This is fine because these events happen with at most negligible probability. Now, we define the following hybrid experiment. Let M^{V^*} be a machine who is given a Hamiltonian cycle to the graph G and works as follows. M follows the strategy of S exactly with the following exception. When M reaches part 2 of the proof for a satisfied execution, it uses its knowledge of the witness to prove the proof in the same way as the honest prover (rather than using its knowledge of the query string q as S does).

We first claim that the distribution generated by M^{V^*} is *identical* to the distribution generated in a real execution between P and V^*. This can be seen by separately considering segments of the transcripts between fixed points. Clearly, all such segments for which S (and therefore M) does

no rewinding are identical. However, notice that even when M does rewind (i.e., in the case that an iteration closes), the final transcript of that segment is according to the random coins used by M before any rewinding took place. Therefore, the transcript is exactly the same as for an honest prover who does no rewinding.

Next, we consider the difference between the distribution generated by M^{V^*} and the distribution generated by S. The difference between these distributions is with respect to the values committed to in the prover commitments of part 2 of the proof. Specifically, M always commits to a random permutation of the graph whereas S sometimes commits to a graph containing a simple cycle only (see Appendix A). Thus, the indistinguishability of the distributions reduces to the computational hiding of the prover-commitments in part 2 of the protocol. The actual reduction is straightforward and is therefore omitted. ■

This completes the proof of Proposition 5. ■

On the length of the preamble. The number of iterations in the preamble is chosen so that all executions are guaranteed to be satisfied before part 2 of the proof is reached; this is demonstrated in Claim 4.4. Specifically, it holds that the number of iterations needs to be greater than or equal to the number of *fixed points* that are placed by the simulator. In the context of bounded concurrent zero-knowledge, this number of fixed points depends only on the number of concurrent executions that take place. We stress that we can extend the preamble to be of any polynomial length (greater than or equal to $2m$), and the number of fixed points required remains $2m$ only. This will become important when Protocol PRS is used for obtaining bounded-concurrent secure two-party computation.

We also remark that the length of the preamble can be reduced to m (rather than $2m$). This is achieved by setting a fixed point only when an execution is satisfied, and not when verifier commitments are sent. Once a potential iteration closes, the simulator rewinds only to the point at which that iteration opened. Nevertheless, we chose to have a preamble of length $2m$ because this makes the analysis simpler (which we believe to be preferable for this work).

On the choice of the [35] protocol. There are currently two known protocols that solve the problem of concurrent zero-knowledge in the plain model and with no timing assumptions. The first is that of Richardson and Kilian [37] and the second is that of Prabhakaran et al. [35] that we use here. The reason why we chose to use the protocol of [35] rather than that of [37] is that it allows for a simpler and cleaner analysis. First, the simulation strategy of [35] is such that the simulator generates exactly the same distribution in every rewinding (see the proof of Claim 4.2 above and Footnote 16 therein). This is in contrast to the [37] protocol where there is a negligible difference between the distributions generated in the first pass and in later rewindings. This negligible difference introduces technical difficulties which are solvable (as shown in [23]), but nevertheless complicate the analysis. Another (secondary) advantage of the [35] proof is that it is possible to have the simulator output the transcript based on what it saw in the first pass (see the description of the simulator in the proof above). Thus it is easy to justify that the output of the simulator is indistinguishable from the output generated in a real proof. (We remark that in the proof of [37], the simulator *must* output the transcript based on what it saw in a later rewinding.)

4.3 Non-Black-Box Bounded Concurrent ZK

In this section, we describe the bounded concurrent zero-knowledge protocol of Barak [1]. The high-level outline of the construction of [1] is as follows. In the first part of the protocol, the

prover and verifier run a "generation protocol" with the following property. A simulator is able to obtain some trapdoor information during the generation protocol that will later enable it to simulate the proof. In contrast, in a real execution between the prover and verifier, the prover has only a negligible chance of obtaining this trapdoor information. Then, the second part of the zero-knowledge protocol is designed such that given the trapdoor information, it is possible to complete the proof (without knowing any witness to the statement being proved). Thus, the simulator works by obtaining the trapdoor information in the first part and then using it to complete the proof in the second part. In contrast, in a real interaction the prover will not obtain the trapdoor information and thus its proof must be based on the validity of the statement being proved.

In this section, we will only describe the generation protocol of the first part of the protocol of [1]. This suffices for our purposes here. For details regarding the second part, which uses universal arguments, see [1, 3].

Easy-hard relations. The idea behind an easy-hard relation is that in a real interaction between the prover and verifier, it is hard for the prover to "hit" the relation. In contrast, it is easy for the simulator to "hit" the relation. Thus, such a relation can be used for part 1 of the proof as described above. Formally,

Definition 6 (easy-hard $\mathsf{Ntime}(n^{\log\log n})$ relations): *Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be an $\mathsf{Ntime}(n^{\log\log n})$ binary relation. We say that R is an* easy-hard relation[17] *if there exist two interactive probabilistic polynomial-time* generating *algorithms P and V that satisfy the following two properties:*

1. Hardness: *For any (non-uniform) polynomial-time machine P^*, if τ is the transcript of the interaction of P^* and V on input 1^n, then the probability that P^* outputs σ such that $(\tau, \sigma) \in R$ is negligible in n.*

2. Easiness: *There exists a polynomial-time computable transformation that takes any (non-uniform) polynomial-time machine V^* and outputs a (non-uniform) probabilistic polynomial-time simulator S^* such that S^* outputs a pair (v, σ) and:*

   (a) *The first string v is computationally indistinguishable from the actual view of V^* when interacting with P. (The view of a party consists of the contents of its tapes and the messages that it receives.)*

   (b) *Let τ be the transcript that is derived from v (note that the transcript of an execution can be derived deterministically from either party's view). Then, except with negligible probability, $(\tau, \sigma) \in R$.*

The pair (P, V) is called a generation protocol for *R.*

The witness σ for τ is the trapdoor information that was mentioned in the motivating discussion above. The "hardness" requirement says that a real prover will not be able to obtain such a σ; whereas the "easiness" requirement says that the simulator will obtain a σ as required. In the second part of the argument system of [1], the prover essentially proves that either the original statement is true or that it knows σ from the generation protocol.

We remark that we have chosen to denote the generating algorithms by P and V as they are played by the prover and verifier in the zero-knowledge protocol. However, this notion can be

[17] Such a relation is called interactive strong easy-hard in [1]. For the sake of brevity, we drop "interactive" and "strong" from the name.

defined independently of zero-knowledge (and indeed, the above is independent of the statement being proved).

Barak showed that assuming the existence of hash functions that are collision-resistant against $n^{\log\log n}$-time adversaries, it suffices to prove the existence of an $\mathsf{Ntime}(n^{\log\log n})$ easy-hard relation in order to derive zero-knowledge. Furthermore, if the hardness and easiness properties hold in the bounded concurrent model, then bounded concurrent zero-knowledge is obtained [1]. We note that the hardness assumption regarding the hash functions has been reduced so that it suffices for them to be collision resistant against any polynomial-time adversary [3]. However, this requires some special properties in the construction of the easy-hard relation as well. We ignore these technicalities and just remark that the weaker hardness assumption suffices for the results presented here. We now proceed to describe the easy-hard relation of [1] for achieving bounded concurrent zero-knowledge.

The Barak easy-hard relation: The main idea behind the relation is to have the transcript τ contain a "proof" that P knows the next message function of V. The hardness property follows from the fact that P^* cannot know V's next message function (and in particular, P^* cannot guess V's random-tape). In contrast, the simulator S^* *does* hold V^*'s next message function and can therefore succeed in generating this "proof". Of course, the only question remaining is what does it mean for P to "prove" that it holds the next message function of V? This was solved by [1] in the following way: P sends V a commitment c to a machine M, and V replies with some random string r. Then, we say that P knows V's next message function if $M(c) = r$ (i.e., when applying the committed machine M to the commitment value c, the output is exactly the message sent by V upon receiving the commitment value c). The key point behind the hardness is that if r is of high enough entropy, then P^* cannot succeed in committing to the correct M. In contrast, since the simulator S^* actually holds the code of V^*, it can just set $M = V^*$ (including the contents of all of V^*'s tapes). Of course, in this case it indeed commits to the correct next message function.

The above works for the stand-alone case. However, in the bounded concurrent model, V^* will not necessarily respond immediately with r. In particular, it may send messages belonging to other executions first. Therefore, it will not hold that $M(c) = r$. The solution to this problem is to say that P holds the next message function if there exists a *"short"* string y such that $M(c, y) = r$. By "short", we mean that y should be at least n bits shorter than the random string r sent by V. The length of r is fixed to be some polynomial $\ell(n)$, where $\ell(n)$ bounds the length of *all* messages received by V during all the concurrent executions. We now describe why this fulfills the requirements of being an easy-hard relation.

- *Hardness:* In order for a cheating P^* to succeed, it must be that $M(c, y) = r$. However, M and c are fixed before V chooses r. Furthermore, $|r| \geq |y|+n$. Therefore, the Kolmogorov complexity of r is too high for it to be predicted by y, and P^* can "hit" the relation with at most negligible probability.
- *Easiness:* As in the stand-alone case, the simulator S^* commits to $M = V^*$. However, here S^* must also determine the string y. This string y is set to be all the P-messages sent by S^* from the time that S^* committed to M (in this execution) until the time that V^* replies with r (in this execution). Since $M = V^*$, it turns out that indeed $M(c, y) = r$. Notice also that since the total length of all P-messages in all concurrent execution is less than $\ell(n)$, it is possible for S^* to define y in this way. Thus, the "easiness" requirement holds.

For more details, see [1]. The generation protocol and easy-hard relation of Barak are as follows:

Construction 1 (Barak generation protocol for an easy-hard relation R):

- **Common input:** 1^n
- **Length parameter:** $\ell(n)$
- **The protocol:**
  1. *V chooses $h \in_R \{0,1\}^n$ (where h defines a collision-resistant hash function with range in $\{0,1\}^n$) and sends it to P.*
  2. *P sends V a commitment $c = C_n(0^n)$.*
  3. *V chooses $r \in_R \{0,1\}^{\ell(n)}$ and sends r to P.*

We now define the easy-hard relation R for which the above is a generation protocol. Let R be the relation of pairs $\{((h,c,r),(M,s,y))\}$ for which the following holds:

1. M is a program of length at most $n^{\log\log n}$ where $n \stackrel{\text{def}}{=} |h|$.
2. c is a commitment to a hash of M using coins s. That is, $c = C_n(h(M);s)$. Note that $|h(M)| = n$.
3. $\mathcal{U}(M,c,y)$ outputs r within $n^{\log\log n}$ steps, where $\mathcal{U}$ is a universal Turing machine.
4. $|y| \leq |r| - n$.

Barak has shown that Construction 1 is a generation protocol in the bounded concurrent model for the easy-hard relation R, as long as $\ell(n)$ (that determines the length of r) fulfills the following requirement: *The total length of all messages sent by P in all executions is less than $\ell(n) - n$.* Thus, both the soundness and zero-knowledge properties of the protocol of [1] hold as long as this condition is fulfilled.

We note that the length of the messages sent by P in part 2 of the zero-knowledge protocol of [1] can be bound by n^2. Since P only sends n bits in the generation protocol, we can bound the total size of all P-messages in the protocol of [1] by $n^2 + n$. Therefore, for m concurrent executions, one can set $\ell(n) = 2mn^2$ and it is guaranteed that r is long enough.

5 A Special-Purpose Composition Theorem

In this section, we define a model whereby the parties run a protocol while being given access to an ideal zero-knowledge (proof of knowledge) functionality. We then present a composition theorem for this model. This theorem is the central tool in our construction of m-bounded concurrent secure two-party computation.

5.1 The ZK-Hybrid Model

In the ZK-hybrid model, all parties have access to many concurrent copies of the following ideal zero-knowledge functionality, parameterized by an NP-relation R:

$$((v,w),\lambda) \mapsto (\lambda,(v,R(v,w)))$$

That is, the first party (the prover) sends a statement v and a witness w to the trusted party computing the functionality. The second party (the verifier) then receives the statement v and a bit b signalling whether or not the prover provided a valid witness for v. We note that this

functionality is actually a proof of knowledge. In order to model a situation where the verifier may reject a proof of even a true statement, we define a special symbol $\perp$ such that for *every* relation R and every v, it holds that $(v, \perp) \notin R$.

Let $p(n)$ be a polynomial and let $\overline{x}, \overline{y} \in \{0,1\}^{p(n)}$ and $z \in \{0,1\}^*$. Then, we denote by $\text{ZK-HYBRID}^m_{\Pi,\mathcal{A}}(\overline{x}, \overline{y}, z)$ the output distribution of $p(n)$ concurrent executions of protocol Π in the ZK-hybrid model, where the scheduling is according to what is allowed in the model of m-bounded concurrency. We stress that the only difference between this model and the real model, is that the zero-knowledge proofs are provided using "trusted help".

5.2 Motivation for the Composition Theorem

Informally speaking, the composition theorem states that a real-model protocol Π can be constructed from any ZK-hybrid model protocol Π' so that the following holds: for any real model adversary $\mathcal{A}$ running Π there exists a ZK-hybrid model adversary $\mathcal{A}'$ running Π' so that the output distributions in both cases are essentially the same.

The basic idea behind the theorem is to construct Π from Π' by replacing the ideal calls to the zero-knowledge functionality with concurrent zero-knowledge protocols. However, two main problems arise when attempting to implement such a strategy:

1. *Malleability/Soundness:* Assume that in the protocol Π' both parties prove zero-knowledge proofs (as is indeed the case for all known protocols for secure two-party computation). Then, we must ensure that the adversary cannot maul the zero-knowledge proof of the honest party in order to cheat. On a technical level this problem arises because the simulation of a zero-knowledge proof is actually "cheating". Therefore, we need to prevent the adversary from utilizing our cheating/simulation in order to cheat in its proofs. In short, the problem here is of soundness and we remark that a naive instantiation of the ideal calls with zero-knowledge protocols would definitely suffer from this problem.

2. *Simulation under concurrency with other protocols:* In the setting of concurrent zero-knowledge, the protocol in question is run concurrently to itself only. In contrast, here the zero-knowledge protocol is run concurrent to itself *and* to other protocols. Now, the natural way to construct the ZK-hybrid model adversary $\mathcal{A}'$ from the real model adversary $\mathcal{A}$ is to have $\mathcal{A}'$ simulate the zero-knowledge proofs for $\mathcal{A}$, while all the other messages (i.e., the messages from the ZK-hybrid model protocol Π') are forwarded to the external honest party. This causes a problem because messages from the zero-knowledge proofs in some executions may be interleaved with messages of Π' in other executions. Since the messages of Π' are forwarded by $\mathcal{A}$ to an external party, $\mathcal{A}$ cannot rewind $\mathcal{A}'$ at these points. This may conflict with the necessity to rewind in order to simulate the zero-knowledge (if the simulation strategy is indeed based on rewinding).

We solve the first problem by using different zero-knowledge protocols so that the adversary cannot maul one protocol in order to cheat in the other. Specifically, we have one of the parties provide proofs using the (non black-box) zero-knowledge protocol of [1] (see Section 4.3), while the other party provides proofs using the modified (black-box) zero-knowledge protocol of [35] (see Section 4.2). Loosely speaking, it turns out that these protocols are both non-malleable with respect to each other. That is, both of the protocols remain sound even if the adversary proves one protocol while simultaneously receiving a *simulated* proof of the other.

The second problem is solved as follows. First, the protocol of [1] does not use rewinding for the simulation. Therefore, no problem arises. (We remark that there are parameters for the

protocol of [1] that must be modified in this scenario, but this is reasonably straightforward.) In contrast, the protocol of [35] does use rewinding. We solve this problem by using the simulation strategy demonstrated in Section 4.2. Recall that this strategy is almost "straight-line". That is, the simulator deals sequentially with each execution and places fixed points behind which it never needs to rewind. The simulation succeeds as long as the number of iterations in the preamble is greater than or equal to the number of fixed points that are placed. Thus, we increase the length of the preamble so that the simulator can place a fixed point for every Π'-message that is sent. This ensure that no rewinding behind a Π'-message is necessary. This strategy is based on the methodology used by [24] to solve the problem of simulating zero-knowledge while external messages may be sent.

5.3 The Composition Theorem

In this section we show that any protocol Π' that runs in the ZK-hybrid model can be transformed into a protocol Π in the real model. The transformation is such that any real-model adversary $\mathcal{A}$ for Π can be simulated by a ZK-hybrid model adversary $\mathcal{A}'$ for Π' in the setting of m-bounded concurrency. We remark that the protocol Π depends both on Π' and on the concurrency bound m.

An important technicality of the transformation is that the ZK-hybrid model adversary $\mathcal{A}'$ runs in expected polynomial-time, irrespective of the distribution over the Π'-messages that it receives. That is, even if the distribution over the Π'-messages that it receives could not be generated by any machine, $\mathcal{A}'$ still remains expected polynomial-time.[18] This will be used later in order to ensure that the ideal-model adversary (obtained in Section 6) runs in expected polynomial-time.

Theorem 7 *Let Π' be a two-party protocol for the* ZK-hybrid model *and let $m(\cdot)$ be polynomial in n. Then, assuming the existence of enhanced trapdoor permutations and collision resistant hash functions, there exists a protocol Π for the real model with the following property. For every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ running Π in the real model, there exists a non-uniform probabilistic expected polynomial-time adversary $\mathcal{A}'$ running Π' in the* ZK-hybrid *model such that for every polynomial $p(n)$,*

$$\left\{\text{ZK-HYBRID}^{m}_{\Pi',\mathcal{A}'}(\overline{x},\overline{y},z)\right\}_{n\in\mathsf{N};\overline{x},\overline{y}\in(\{0,1\}^n)^{p(n)};z\in\{0,1\}^*} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}^{m}_{\Pi,\mathcal{A}}(\overline{x},\overline{y},z)\right\}_{n\in\mathsf{N};\overline{x},\overline{y}\in(\{0,1\}^n)^{p(n)};z\in\{0,1\}^*}$$

Furthermore, the expected running-time of $\mathcal{A}'$ is polynomial for every distribution over the messages of Π' that it may receive.

Proof: We begin by showing how to construct Π. Denote the participating parties by P_1 and P_2. Then, both P_1 and P_2 may prove zero-knowledge proofs in Π'. Protocol Π is obtained by *replacing* all the ideal zero-knowledge calls of Π' with the zero-knowledge arguments of knowledge of [4], described in Section 4.1. We denote this protocol by POK. Recall that in this argument system, both the prover and verifier prove a statement in zero-knowledge; we call these the subproofs. Now, all the zero-knowledge subproofs in which P_1 is the prover are proven using the PRS zero-knowledge protocol of Section 4.2. Furthermore, all the zero-knowledge subproofs in which P_2 is the

[18] This would be trivial if $\mathcal{A}'$ was strict polynomial-time. However, an expected polynomial-time interactive machine is only required to be expected polynomial-time when it interacts with another machine, and thus when it receives messages distributions that are generated through such interactions. Therefore, there may be message distributions in which such a machine always runs an exponential number of steps. Of course, such message distributions would not be obtainable through real interaction, but may be generated in simulation. See [32, Appendix A.2] for such an example.

prover are proven using the concurrent zero-knowledge protocol of [1], denoted BZK, as described in Section 4.3. That is, when P_1 plays the prover in the zero-knowledge proof of knowledge POK, then P_2 provides a subproof (as the receiver in the commit-with-extract scheme of part 1) using Protocol BZK and P_1 provides a subproof (proving the correctness of its commitment in part 1) using Protocol PRS. Conversely, when P_2 plays the prover in the zero-knowledge proof of knowledge POK, then P_1 provides a subproof (as the receiver in the commit-with-extract scheme) using Protocol PRS and P_2 provides a subproof (proving the correctness of its commitment in part 1) using Protocol BZK. *The important thing to remember is that all the subproofs going from P_1 to P_2 use* PRS *and all the subproofs going from P_2 to P_1 use* BZK. We note that the proof of knowledge POK and the subproofs PRS and BZK can all be implemented assuming the existence of enhanced trapdoor permutations and collision resistant hash functions.

Before continuing, we formally define what it means to "replace" an ideal zero-knowledge call with a protocol:

- *Prover:* If a party is instructed in Π' to send (v, w) to the trusted party computing the zero-knowledge functionality, then it plays the prover in the zero-knowledge proof of knowledge protocol, using the witness w that it has. The messages that it sends and receives within this proof are not recorded on its transcript for Π'.
- *Verifier:* If in Π' a party is to receive a message (v, b) from the trusted party computing the zero-knowledge functionality, then it plays the verifier in the zero-knowledge proof of knowledge protocol. If it accepts the proof, then it writes $(v, 1)$ on its transcript for Π'. Otherwise it writes $(v, 0)$. As above, the messages within the proof are not recorded on its transcript for Π'.

As we have described above, the PRS and BZK protocols are used as subproofs in POK. Both of these protocols are parameterized: the PRS protocol by the number of iterations k in the preamble and the BZK protocol by the length $\ell(n)$ of the verifier message r. Recall that in the scenario of m-bounded concurrent zero-knowledge, in PRS the parameter k is set to $2m$, and in BZK the parameter $\ell(n)$ is set to $2mn^2$. Now, we denote by $\mathsf{rounds}(\mathsf{Prot})$ the number of rounds in Protocol Prot and by $\mathsf{length}(\mathsf{Prot})$ the total length (i.e., number of bits) of all messages sent in Protocol Prot. Furthermore, we denote by ζ the total number of ideal calls made by both parties to the ideal zero-knowledge functionality in Π'. Finally, when we refer to $\mathsf{rounds}(\mathsf{POK})$ and $\mathsf{length}(\mathsf{POK})$, we mean the number of rounds and the length of the proof of knowledge *without* the subproofs PRS and BZK. The parameters for the zero-knowledge subproofs are set as follows: In PRS we set

$$k = 2\zeta m + m \cdot \Big(\mathsf{rounds}(\Pi') + \zeta \cdot \mathsf{rounds}(\mathsf{POK}) + \zeta \cdot \mathsf{rounds}(\mathsf{BZK})\Big)$$

and in BZK we set

$$\ell(n) = 2\zeta m n^2 + m \cdot \Big(\mathsf{length}(\Pi') + \zeta \cdot \mathsf{length}(\mathsf{POK}) + \zeta \cdot \mathsf{length}(\mathsf{PRS})\Big)$$

In other words, the number of iterations in the PRS preamble is $2\zeta m$ (there are ζm executions of PRS and 2 iterations are needed in the preamble for each of these executions), plus m times the combined number of rounds in Π excluding the PRS subproofs; this combined number of rounds comes to $\mathsf{rounds}(\Pi') + \zeta \cdot \mathsf{rounds}(\mathsf{POK}) + \zeta \cdot \mathsf{rounds}(\mathsf{BZK})$.

Likewise, the length of the message r in BZK is set to $2\zeta m n^2$ (as in ζm-bounded concurrent zero-knowledge) plus m times the length of all other messages sent. This ensures that r is at least n bits longer than all the messages sent by P_2 in all m executions. We remark that there is no problem setting the parameters in such a way. That is, although $\ell(n)$ is dependent on $\mathsf{length}(\mathsf{PRS})$, the number of rounds in BZK is *not* dependent on PRS. Thus, we can first set k and then $\ell(n)$.

We digress here for a moment and comment that in our final protocol, $\mathsf{rounds}(\Pi')$ and ζ are both constants and thus the final number of rounds in Π is $O(m)$.

We now show that for every real model adversary $\mathcal{A}$ running Π, there exists a ZK-hybrid model $\mathcal{A}'$ running Π' such that the output distributions generated by $\mathcal{A}$ and $\mathcal{A}'$ are indistinguishable. Intuitively, this is because for m-bounded concurrency the zero-knowledge arguments of knowledge as described constitute "good" replacements for the ideal calls to the zero-knowledge functionality. Loosely speaking, we construct $\mathcal{A}'$ from $\mathcal{A}$ as follows. $\mathcal{A}'$ internally incorporates $\mathcal{A}$ and externally forwards all the Π'-messages (i.e., the messages not belonging to the zero-knowledge proofs) between $\mathcal{A}$ and the external party with which $\mathcal{A}'$ interacts. In contrast, $\mathcal{A}'$ internally simulates the zero-knowledge proofs that $\mathcal{A}$ expects to see. Specifically, upon receiving a message $(v, 1)$ from the trusted party computing the ideal zero-knowledge functionality, $\mathcal{A}'$ simulates the zero-knowledge proof of knowledge for v, where $\mathcal{A}$ plays the verifier. Likewise, when $\mathcal{A}$ proves a proof of knowledge for a statement v, the ZK-hybrid model adversary $\mathcal{A}'$ runs the extractor for the proof. If $\mathcal{A}'$ obtains a valid witness w, then it sends (v, w) to the trusted third party; otherwise, it sends $(v, \perp)$. Such a strategy ensures that the output distribution generated by $\mathcal{A}'$ in a run of Π' (in the ZK-hybrid model) is indistinguishable from the output distribution generated by $\mathcal{A}$ in a run of Π (in the real model). Of course, the main challenge (and one of the central contributions of this work), is in showing how to carry out the simulation and extraction when the zero-knowledge proofs of knowledge are subprotocols within a larger protocol that is executed in the setting of m-bounded concurrency. We stress that it does not suffice to prove the correctness of the simulation and extraction within the context of m-bounded concurrent zero-knowledge. This is because now the zero-knowledge proof is run concurrently with Π', and not only concurrently with itself. We now formally prove the existence of a ZK-hybrid model adversary $\mathcal{A}'$ for every real model adversary $\mathcal{A}$. We separately deal with the cases that P_1 and P_2 are corrupted.

Party P_1 is corrupted. Let $\mathcal{A}$ be a real-model adversary for Π. We construct an adversary $\mathcal{A}'$ who internally incorporates $\mathcal{A}$ and works in the ZK-hybrid model interacting with P_2 in protocol Π'. Adversary $\mathcal{A}'$ sends $\mathsf{internal}$ and $\mathsf{external}$ messages. Internal messages are sent by $\mathcal{A}'$ to the internally incorporated $\mathcal{A}$, whereas external messages are sent by $\mathcal{A}'$ to the external honest party P_2 and to the external trusted party computing the ideal zero-knowledge functionality. As we have mentioned, all the Π'-messages are defined to be external, whereas all of the messages belonging to POK are defined to be internal. Thus, except for the ideal zero-knowledge calls, $\mathcal{A}'$ forwards all of the Π'-messages unmodified between $\mathcal{A}$ and P_2. In addition:

- Whenever $\mathcal{A}$, controlling P_1, proves a zero-knowledge proof of knowledge for some statement v, $\mathcal{A}'$ runs the *extraction* strategy for POK. (Actually, what is needed is witness-extended emulation; see [32, 4] for details.) This involves simulating Protocol BZK for $\mathcal{A}$ as the verifier (because the subproof of part 1 that is simulated is given by P_2 to P_1) and honestly verifying Protocol PRS where $\mathcal{A}$ is the prover (because the subproof of part 2 is given by P_1). (In addition, the receiver messages of the commit-with-extract scheme of part 1 are chosen according to the extraction strategy; see Section 4.1.) From this extraction procedure, $\mathcal{A}'$ receives some string w (which is "hopefully" a witness for v). $\mathcal{A}'$ verifies that $(v, w) \in R$. If yes, then $\mathcal{A}'$ externally sends (v, w) to the trusted party computing the ideal zero-knowledge functionality. Otherwise, it sends $(v, \perp)$.
- Whenever $\mathcal{A}'$ receives an external message $(v, 1)$ from the trusted party, it internally *simulates* the zero-knowledge proof of knowledge POK for $\mathcal{A}$. This involves honestly verifying Protocol PRS where $\mathcal{A}$ is the prover (from part 1 of POK) and simulating BZK for $\mathcal{A}$ as the verifier (from

part 2 of POK). (In addition, the value committed to by $\mathcal{A}'$ in the commit-with-extract scheme of part 1 is garbage; see Section 4.1.)
(We note that $\mathcal{A}'$ never receives a message $(v, 0)$ from the trusted party because P_2 is honest and we assume that honest provers always check that they have a correct witness before they send anything to the trusted party or attempt to prove POK.)

The simulation of protocol BZK is carried out using the simulator of [1], as described in Section 4.3. Having described the simulation by the ZK-hybrid model adversary $\mathcal{A}'$, we proceed to prove that it generates a distribution that is indistinguishable from a real model execution.

PROOF OUTLINE. Intuitively, the simulation strategy by $\mathcal{A}'$ works as long as Protocol PRS is sound and the simulator of BZK succeeds in generating a view that is indistinguishable from a real execution of BZK. However, in order to prove this, we need to define a hybrid experiment that isolates the PRS and BZK executions. We do this by defining a weaker ideal zero-knowledge functionality that is not a proof of knowledge. We then show that replacing the PRS and BZK subproofs by this ideal functionality makes at most a negligible difference. This enables us to deal with the PRS and BZK subproofs separately.

THE HYBRID EXPERIMENT. The hybrid experiment involves a (weaker) ideal zero-knowledge functionality, parameterized by an NP-language L and defined by the following:

$$(v, \lambda) \rightarrow (\lambda, (v, \chi_L(v)))$$

where $\chi_L(v) = 1$ if and only if $v \in L$. That is, the proving party sends a statement v to the trusted party, who then sends $(v, 1)$ to the verifying party if $v \in L$, and $(v, 0)$ if $v \notin L$. In order to reflect the possibility that a prover may fail to prove a correct statement (in a protocol execution that replaces this ideal functionality), we define that if v has the symbol $\perp$ appended to it (i.e., if it is of the form $v \circ \perp$), then the trusted party sends $(v, 0)$ to the verifier irrespective of whether or not $v \in L$. We note that this ideal functionality may not be efficient (in particular, if L is a hard language, then verifying whether or not $v \in L$ may take super-polynomial time); nevertheless this suffices here. We denote this hybrid experiment by WEAKZK-HYBRID. (There is nothing really weak about this ideal functionality except that it is *not* a proof of knowledge.)

STEP 1 – REPLACING BZK WITH (WEAK) IDEAL ZERO-KNOWLEDGE. We now define a protocol $\hat{\Pi}$ that works in the weakZK-hybrid model. $\hat{\Pi}$ is the same as the real-model protocol Π except that instead of P_2 proving a statement v to P_1 using the subproof BZK, it sends v to the trusted party computing the ideal weak zero-knowledge functionality. Subproofs given by P_1 using PRS are unmodified.

We now show that there exists an adversary $\hat{\mathcal{A}}$ for $\hat{\Pi}$ that interacts with P_2 in the weakZK-hybrid model and who generates an output distribution that is indistinguishable from an execution of $\mathcal{A}$ with Π in the real model. Notice that the only difference between Π and $\hat{\Pi}$ is whether proofs given by P_2 (who in this case is honest) are carried out by running BZK or by using the weak zero-knowledge functionality. Thus, the proof of indistinguishability is based on the zero-knowledge property of BZK in this scenario.

The adversary $\hat{\mathcal{A}}$ for $\hat{\Pi}$ works as follows. $\hat{\mathcal{A}}$ internally incorporates $\mathcal{A}$ and externally forwards all Π', POK and PRS messages between $\mathcal{A}$ and P_2. However, messages of BZK are internally simulated for $\mathcal{A}$ by $\hat{\mathcal{A}}$. That is, whenever $\hat{\mathcal{A}}$ receives a message $(v, 1)$ from the trusted party computing the

ideal weak zero-knowledge functionality, it simulates an execution of BZK with $\mathcal{A}$ as the verifier. We now prove that

$$\left\{\text{WEAKZK-HYBRID}^{m}_{\hat{\Pi},\hat{\mathcal{A}}}(\overline{x},\overline{y},z)\right\} \stackrel{c}{\equiv} \left\{\text{REAL}^{m}_{\Pi,\mathcal{A}}(\overline{x},\overline{y},z)\right\} \tag{1}$$

The difference between an execution of Π with $\mathcal{A}$ and an execution of $\hat{\Pi}$ with $\hat{\mathcal{A}}$ is that in Π party P_2 proves real BZK proofs, whereas in $\hat{\Pi}$ they are simulated. Thus, as we have mentioned Eq. (1) follows from the zero-knowledge property of BZK. It remains to show that the BZK simulation "works" in this scenario (i.e., within $\hat{\Pi}$). However, as we have discussed in Section 4.3, the simulator for BZK works as long as the total number of messages sent by the prover (here P_2) in all the executions of Π is less than $\ell(n) - n$. However, this follows immediately from the way $\ell(n)$ was chosen.

Step 2 – replacing PRS with (weak) ideal zero-knowledge. We now define a protocol $\tilde{\Pi}$ which is the same as Π except that *both* the subproofs PRS and BZK are proven using calls to the ideal weak zero-knowledge functionality. Thus, the only difference between $\hat{\Pi}$ and $\tilde{\Pi}$ is whether proofs given by P_1 (who in this case is controlled by the adversary) to P_2 are given by running PRS or by using the weak zero-knowledge functionality. Thus, the proof of indistinguishability is based on the *soundness* of PRS in this scenario.

We define an adversary $\tilde{\mathcal{A}}$ who works in exactly the same way as $\hat{\mathcal{A}}$ except that it internally verifies the zero-knowledge subproofs of PRS provided by $\hat{\mathcal{A}}$. Formally, $\tilde{\mathcal{A}}$ internally incorporates $\hat{\mathcal{A}}$ and externally forwards all Π', POK and ideal weak zero-knowledge calls that already existed in $\hat{\Pi}$. However, messages of PRS are dealt with internally. That is, whenever $\hat{\mathcal{A}}$ proves a statement v using PRS, $\tilde{\mathcal{A}}$ internally verifies the proof (playing the honest verifier strategy). If $\tilde{\mathcal{A}}$ accepts the proof, then it sends v to the trusted party computing the ideal weak zero-knowledge functionality. Otherwise, it sends $v \circ \perp$ to the trusted party. We prove that

$$\left\{\text{WEAKZK-HYBRID}^{m}_{\tilde{\Pi},\tilde{\mathcal{A}}}(\overline{x},\overline{y},z)\right\} \stackrel{c}{\equiv} \left\{\text{WEAKZK-HYBRID}^{m}_{\hat{\Pi},\hat{\mathcal{A}}}(\overline{x},\overline{y},z)\right\} \tag{2}$$

First, assume that $\tilde{\mathcal{A}}$ *never* accepts a proof v from $\hat{\mathcal{A}}$ when $v \notin L$. Under this assumption, the outputs of $\hat{\mathcal{A}}$ and P_2 are *identical* in $\hat{\Pi}$ and $\tilde{\Pi}$ (recall that P_2's output is determined by its Π'-transcript only). This can be seen as follows. First, when $\tilde{\mathcal{A}}$ verifies PRS subproofs from $\hat{\mathcal{A}}$ in $\tilde{\Pi}$, it plays the honest verifier, exactly like P_2 in $\hat{\Pi}$. Therefore, $\hat{\mathcal{A}}$'s view is the same in both cases. Furthermore, the probability that $\tilde{\mathcal{A}}$ accepts a proof from $\hat{\mathcal{A}}$ in $\tilde{\Pi}$ is exactly the same as the probability that P_2 accepts a proof from $\hat{\mathcal{A}}$ in $\hat{\Pi}$. Now, if $\tilde{\mathcal{A}}$ accepts a proof from $\hat{\mathcal{A}}$, then P_2 receives $(v,1)$ in $\tilde{\Pi}$. This is because in this case $\tilde{\mathcal{A}}$ sends v to the trusted party computing the ideal weak zero-knowledge functionality, and by the above assumption, it must hold that $v \in L$. The trusted party therefore sends $(v,1)$ to P_2. Likewise, if P_2 accepts a proof from $\hat{\mathcal{A}}$ in $\hat{\Pi}$, then it writes $(v,1)$ on its Π'-transcript. Similarly, when $\tilde{\mathcal{A}}$ rejects a proof in $\tilde{\Pi}$, it sends $v \circ \perp$ to the trusted party and P_2 receives $(v,0)$ (irrespective of whether or not $v \in L$). However, this is exactly what P_2 would write on its transcript in $\hat{\Pi}$ if it rejected a proof from $\hat{\mathcal{A}}$. We conclude that the output distribution of $\tilde{\mathcal{A}}$ and P_2 in $\tilde{\Pi}$ is identical to the output distribution of $\hat{\mathcal{A}}$ and P_2 in $\hat{\Pi}$.

The above argument assumes that $\tilde{\mathcal{A}}$ never accepts a proof from $\hat{\mathcal{A}}$ for which $v \notin L$. Thus, in order to prove Eq. (2), it suffices to show that the *soundness* of PRS holds here. That is, we show that the probability that $\tilde{\mathcal{A}}$ accepts a proof when $v \notin L$ is negligible in the security parameter n. (Note that if this "bad event" was to occur, then P_2 would write $(v,1)$ on its Π'-transcript in $\hat{\Pi}$ but would receive $(v,0)$ from the trusted party in $\tilde{\Pi}$.) Since the soundness of PRS is unconditional, it suffices for us to construct an all-powerful cheating prover P^*. Specifically, assume by contradiction

that for some set of inputs $\overline{x}, \overline{y}$ and z, adversary $\hat{\mathcal{A}}$ successfully proves a false statement to $\tilde{\mathcal{A}}$ with non-negligible probability. Then, P^* computes the coins of $\hat{\mathcal{A}}$, $\tilde{\mathcal{A}}$ and P_2 which maximize the probability of $\hat{\mathcal{A}}$ successfully proving the false statement; actually, P^* computes the coins for all messages sent by $\tilde{\mathcal{A}}$ except for those used in verifying the false proof from $\hat{\mathcal{A}}$. Then, P^* internally emulates the entire execution with $\hat{\mathcal{A}}$, $\tilde{\mathcal{A}}$ and P_2, except for the PRS subproof of the false statement proven by $\hat{\mathcal{A}}$; the messages of this subproof are sent externally to the verifier V. (Notice that this emulation can be carried out perfectly by P^*, including the ideal weak zero-knowledge calls, because P^* is not computationally bounded.) Now, since $\tilde{\mathcal{A}}$ verifies PRS subproofs from $\hat{\mathcal{A}}$ using the honest verifier strategy and it accepts the false proof with non-negligible probability, the probability that V accepts the false proof from $\hat{\mathcal{A}}$ (as forwarded by P^*) is also non-negligible. This contradicts the soundness of the PRS subproof and thus Eq. (2) follows.

Remark: This soundness argument is facilitated (in part) by the fact that the simulation by $\hat{\mathcal{A}}$ and $\tilde{\mathcal{A}}$ requires no rewinding. Rewinding would cause a problem because then the same verifier message to be sent by V may appear a number of times. However, V is an external party that sends each message only once and cannot be rewound.

STEP 3 – CONCLUSION. The proof of the case that P_1 is corrupted is concluded by showing that the distribution generated by $\tilde{\mathcal{A}}$ in $\tilde{\Pi}$ in the weakZK-hybrid model, is indistinguishable from the distribution generated by $\mathcal{A}'$ in Π' in the ZK-hybrid model. (The strategy of adversary $\mathcal{A}'$ is described above and is the same as $\tilde{\mathcal{A}}$ except for the treatment of POK messages.) That is, we show the following:[19]

$$\left\{\text{ZK-HYBRID}^{m}_{\Pi',\mathcal{A}'}(\overline{x},\overline{y},z)\right\} \stackrel{c}{\equiv} \left\{\text{WEAKZK-HYBRID}^{m}_{\tilde{\Pi},\tilde{\mathcal{A}}}(\overline{x},\overline{y},z)\right\} \tag{3}$$

The difference between an execution of $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$ and Π' with $\mathcal{A}'$ is as follows. In $\tilde{\Pi}$, the messages of POK that do not belong to the zero-knowledge subproofs BZK and PRS are sent by the external party P_2. However, in Π', these messages are internally simulated by $\mathcal{A}'$ for $\tilde{\mathcal{A}}$. Now, the simulation and extraction strategy for POK succeeds as long as the subproofs in which the adversary plays the verifier are "properly" simulated, and as long as soundness is preserved in the subproofs in which the adversary plays the prover; see the brief description above, and [4] for details. The simulation of the subproofs is already taken care of by the fact that $\tilde{\Pi}$ works in the ideal weak zero-knowledge hybrid model. Likewise, by the construction of $\tilde{\mathcal{A}}$, we know that except with negligible probability it only sends v to the trusted party computing the weak ideal zero-knowledge functionality if v is a correct statement (otherwise it sends $v \circ \perp$). Therefore, upon receiving v from $\tilde{\mathcal{A}}$, adversary $\mathcal{A}'$ can assume that v is correct. We conclude that the simulation of POK by $\mathcal{A}'$ in Π' yields a distribution that is indistinguishable from that of $\tilde{\mathcal{A}}$ in $\tilde{\Pi}$. Eq. (3) follows. The theorem for the case that P_1 is corrupt follows by combining Equations (1), (2) and (3).

We now proceed to the case that P_2 is corrupted.

Party P_2 is corrupted. We construct the ZK-hybrid model adversary $\mathcal{A}'$ in a similar way to the previous case. $\mathcal{A}'$ internally incorporates $\mathcal{A}$ and externally forwards all Π'-messages between $\mathcal{A}$ and P_1. In addition:

[19] We stress an important subtlety in our proof here. We do *not* prove that for every adversary $\tilde{\mathcal{A}}$ there exists an adversary $\mathcal{A}'$ for which the output distributions are indistinguishable. Rather, we show this for the specific adversary $\tilde{\mathcal{A}}$ described above. Specifically, it is crucial to our proof that $\tilde{\mathcal{A}}$ would sends a false statement v to the trusted party computing the weak ideal zero-knowledge functionality with at most negligible probability. If this were not the case (like with general adversaries), we would be "stuck" because we may not be able to verify whether or not $v \in L$.

- Whenever $\mathcal{A}$, controlling P_2, proves a zero-knowledge proof of knowledge for some statement v, $\mathcal{A}'$ runs the extraction strategy for POK. This involves simulating Protocol PRS for $\mathcal{A}$ as verifier (because the subproof of part 1 of POK that is simulated is given by P_1 to P_2), and honestly verifying Protocol BZK where $\mathcal{A}$ is the prover (because the subproof of part 2 is given by P_2). (In addition, the receiver messages of the commit-with-extract scheme of part 1 are chosen according to the extraction strategy; see Section 4.1.) From this extraction procedure, $\mathcal{A}'$ receives some string w. $\mathcal{A}'$ verifies that $(v, w) \in R$. If yes, then $\mathcal{A}'$ externally sends (v, w) to the trusted party computing the ideal zero-knowledge functionality. Otherwise, it sends $(v, \perp)$.
- Whenever $\mathcal{A}'$ receives an external message $(v, 1)$ from the trusted party, it internally simulates the zero-knowledge proof of knowledge POK for $\mathcal{A}$. This involves honestly verifying Protocol BZK where $\mathcal{A}$ is the prover (from part 1 of POK) and simulating PRS for $\mathcal{A}$ as the verifier (from part 2 of POK). (In addition, the value committed to by $\mathcal{A}'$ in the commit-with-extract scheme of part 1 is garbage; see Section 4.1.)

We now describe how $\mathcal{A}'$ carries out the simulation of PRS. The simulation uses a very similar strategy to that described in the proof of Proposition 5 (see Section 4.2). In our description here, we will use notation and terminology taken from the proof of Proposition 5 (and we therefore assume familiarity with the details of that proof). $\mathcal{A}'$ interacts with $\mathcal{A}$ in a black-box manner and externally forwards all Π'-messages to P_1. Likewise, replies from P_1 are returned to $\mathcal{A}$. Furthermore, $\mathcal{A}'$ sets a fixed point after any of the following events:

1. A non-PRS message is sent by $\mathcal{A}$; this includes all messages belonging to Π', BZK and POK. (Recall that when we refer to POK, we mean the proof of knowledge not including the BZK and PRS subproofs.)
2. A verifier commitment message from an execution of PRS is sent by $\mathcal{A}$.
3. An execution of Protocol PRS is aborted.
4. A potential iteration closes.

Recall that no rewinding ever takes place behind a fixed point. Now, $\mathcal{A}'$ works in exactly the same way as the simulator S in the proof of Proposition 5. Specifically, whenever a fixed point of type (1), (2) or (3) above is sent, nothing is done by $\mathcal{A}'$ except to fix the transcript until (and including) the current message. (This is analogous to S's behavior when a verifier commitment message is sent.) Furthermore, whenever an execution of PRS is aborted, this execution is satisfied. Finally, whenever a potential iteration closes, $\mathcal{A}'$ repeatedly rewinds $\mathcal{A}$ to the current fixed point until this iteration is once again the first to close (without any other event causing a fixed point to be set occurring). We stress that in each of these repetitions, $\mathcal{A}'$ uses new uniformly chosen random coins. Furthermore, if $\mathcal{A}$ sends a non-PRS message, then $\mathcal{A}'$ does *not* send it externally to P_1 (because this would prevent any further rewinding). Rather, whenever a different event causing a fixed point to be set occurs, $\mathcal{A}'$ immediately rewinds $\mathcal{A}$ to the current fixed point and starts again. Now, when $\mathcal{A}'$ succeeds in having the same potential iteration close a second time, it repeats the execution from the previous fixed point using the *same* random coins that it used the first time (in exactly the same way that S does in the proof of Proposition 5). Recall that at this point, the execution is satisfied with overwhelming probability and $\mathcal{A}'$ knows the query string q. Thus, $\mathcal{A}'$ can successfully simulate part 2 of the proof in this execution of PRS (unless it outputs failure). We remark that $\mathcal{A}'$ outputs failure or unsatisfied based on exactly the same events as S. This completes the description of the simulation. We begin by proving that $\mathcal{A}'$ runs in expected polynomial-time. We start with the following claim.

Claim 5.1 *The expected-time taken by $\mathcal{A}'$ for setting each fixed point is polynomial in n.*

This follows using the same analysis as in Claim 4.1. (The only difference is that there are additional events causing a fixed point to be set – specifically, the sending of a non-PRS message. However, each of these only require a fixed polynomial number of steps and the number of events overall is still polynomial.) In order to conclude the analysis of $\mathcal{A}'$'s running-time, notice that there are only a polynomial number of possible fixed points (bounded by the number of Π' and BZK messages plus twice the number of PRS executions). The fact that $\mathcal{A}'$ runs in expected polynomial-time is therefore derived from Claim 5.1 and the linearity of expectations. However, we actually need to prove that the expected running-time of $\mathcal{A}'$ is polynomial, for *every* distribution over the messages of Π'. This follows from the fact that no rewinding is carried out over Π'-messages. That is, a fixed point is set every time a Π'-message is forwarded by $\mathcal{A}'$ to $\mathcal{A}$. Therefore, the running-time of $\mathcal{A}'$ can be divided into the time that it takes to set the fixed points, plus the time that it takes for $\mathcal{A}$ to reply to the Π'-messages. Above, we have shown that the expected time it takes to set the fixed points is polynomial. Regarding the Π'-messages, since $\mathcal{A}$ runs in strict polynomial-time, these account for a polynomial number of steps, irrespective of the distribution over the messages, as required. We therefore have the following claim:

Claim 5.2 *$\mathcal{A}'$ runs in expected-time that is polynomial in n. Furthermore, this holds for every distribution over the messages of Π' that $\mathcal{A}'$ may receive.*

We remark that in the proof of Proposition 5, the complexity of the verifier V^* was not included in S's running-time (since S is a black-box simulator). In contrast, here we do need to incorporate the running-time of $\mathcal{A}$ into the complexity of $\mathcal{A}'$. However, since $\mathcal{A}$ is a (strict) polynomial-time adversary, this only increases the complexity of $\mathcal{A}'$ by a fixed polynomial. Having established that $\mathcal{A}'$ runs in expected polynomial-time, we proceed to prove that the resulting distribution of $\mathcal{A}'$ in Π' is computationally indistinguishable from the resulting distribution of $\mathcal{A}$ with Π. We prove this using the same hybrid experiment as in the case that P_1 is corrupted.

STEP 1 – REPLACING PRS WITH (WEAK) IDEAL ZERO-KNOWLEDGE. We define the protocol $\tilde{\Pi}$ for the weakZK-hybrid model in the same way as in the case that P_1 is corrupted. Therefore, in this protocol, ideal calls to the weak zero-knowledge functionality are used instead of running the PRS subproofs. We now demonstrate the existence of an adversary $\tilde{\mathcal{A}}$ such that the output distribution of an execution of $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$ is indistinguishable from an execution of Π with $\mathcal{A}$ (in the real model). $\tilde{\mathcal{A}}$ internally incorporates $\mathcal{A}$ and forwards all non-PRS messages between $\mathcal{A}$ and the external P_1. However, when $\tilde{\mathcal{A}}$ receives a message $(v, 1)$ from the trusted party computing the ideal weak zero-knowledge functionality, it simulates a PRS execution for $\mathcal{A}$ using exactly the same strategy described above for $\mathcal{A}'$. We now prove that,

$$\left\{\text{WEAKZK-HYBRID}^{m}_{\tilde{\Pi},\tilde{\mathcal{A}}}(\overline{x},\overline{y},z)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}^{m}_{\Pi,\mathcal{A}}(\overline{x},\overline{y},z)\right\} \tag{4}$$

Proving Eq. (4) involves showing that the simulation by $\tilde{\mathcal{A}}$ is indistinguishable from a real proof by P_1 (this is the only difference between Π and $\tilde{\Pi}$). This follows very similar lines to the proof of Proposition 5. We first claim that $\tilde{\mathcal{A}}$ outputs failure with only negligible probability:

Claim 5.3 *The probability that $\tilde{\mathcal{A}}$ outputs* failure *is negligible.*

The proof of this claim is almost identical to the proof of Claim 4.3. (The only difference is that there are ζm executions of PRS instead of m and the reduction to the computational binding includes emulating all of the Π' and BZK messages as well. However, this emulation can be carried out because the machine attempting to break the binding property of the commitments is given both inputs vectors $\overline{x}$ and $\overline{y}$.) Next, we claim that $\tilde{\mathcal{A}}$ never outputs unsatisfied:

Claim 5.4 *The probability that* $\tilde{\mathcal{A}}$ *outputs* unsatisfied *equals* 0.

Proof: As in the proof of Claim 4.4, it suffices to show that every PRS execution which is not aborted has a potential iteration which is the first to close after some fixed point. This is demonstrated by counting the number of iterations and the number of fixed points, and noticing that at most one iteration can be opened between each two fixed points. Now, the number of fixed points that are set by $\tilde{\mathcal{A}}$ during the simulation equals exactly

$$2\zeta m + m \cdot \Big(\mathsf{rounds}(\Pi') + \zeta \cdot \mathsf{rounds}(\mathsf{POK}) + \zeta \cdot \mathsf{rounds}(\mathsf{BZK})\Big)$$

This number is reached as follows. The number of executions of PRS equals ζm. Thus, there are at most $2\zeta m$ fixed points for all of these executions (ζm for the verifier commitments and ζm for when each one is satisfied). Furthermore, a fixed point is set for each of the non-PRS messages (these messages are from Π', BZK, and POK). For m concurrent executions, we have $m \cdot \mathsf{rounds}(\Pi')$ messages from Π', $m \cdot \zeta \cdot \mathsf{rounds}(\mathsf{POK})$ from POK, and $m \cdot \zeta \cdot \mathsf{rounds}(\mathsf{BZK})$ messages from BZK. Summing these up we obtain the exact number k of iterations in the preamble of PRS.

Now, no unsatisfied execution of PRS can reach part 2 (of PRS) before k fixed points are set. Furthermore, k fixed points can only be set once all executions are satisfied. Thus we conclude that every execution of PRS must be satisfied before part 2 of an unsatisfied PRS subproof is reached. ■

Having established the above, the indistinguishability of the distributions follows with an analogous hybrid argument to the proof of Claim 4.5. Specifically, we define a hybrid adversary/simulator $\tilde{S}$ who receives the witnesses for each proof being proved (from the honest P_1) and runs part 2 of the proof of PRS as P_1 would (i.e., using the witness). However, all other parts of the experiment are run exactly according to the instructions of $\tilde{\mathcal{A}}$ (including the simulation of the preamble of PRS).

Now, the transcript of the PRS messages in the experiment with $\tilde{S}$ is *identical* to the transcript of the PRS messages in Π. This can be seen as follows. The messages sent between fixed points in which no rewinding was done are clearly distributed identically in Π and in the experiment with $\tilde{S}$. (Notice that the messages of part 2 of PRS may be included here. However, $\tilde{S}$ proves part 2 using the same witness as P_1; therefore the distributions are the same.) The same is also true of the messages sent between fixed points in which $\tilde{S}$ does rewind. This can be seen because after the rewinding has concluded, $\tilde{S}$ replays the execution using the same random coins as the first time. Therefore, the distribution is the same as when no rewinding takes place.

Next, we claim that the output distribution in the experiment with $\tilde{S}$ is computationally indistinguishable from the execution of $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$. This follows from the fact that the only difference between the two experiments relates to the commitments used in proving part 2 of the PRS subproofs. This can therefore be reduced to the hiding property of the commitment scheme. This is straightforward and details are thus omitted. This completes the proof of Eq. (4).

STEP 2 – REPLACING BZK WITH (WEAK) IDEAL ZERO-KNOWLEDGE. Again, as in the case that P_1 is corrupted, we define a protocol $\hat{\Pi}$ in which both the subproofs PRS and BZK are proven using the ideal weak zero-knowledge functionality. Furthermore, an adversary $\hat{\mathcal{A}}$ is defined who incorporates

$\tilde{\mathcal{A}}$ and externally forwards all Π'-messages between $\tilde{\mathcal{A}}$ and P_1. However, when $\tilde{\mathcal{A}}$ proves a BZK subproof of a statement v, $\hat{\mathcal{A}}$ verifies this internally, If $\hat{\mathcal{A}}$ accepts the proof, it sends v to the trusted party computing the ideal weak zero-knowledge functionality; otherwise, it sends $v \circ \perp$. We now prove that

$$\left\{\text{WEAKZK-HYBRID}^{m}_{\hat{\Pi},\hat{\mathcal{A}}}(\overline{x},\overline{y},z)\right\} \stackrel{c}{\equiv} \left\{\text{WEAKZK-HYBRID}^{m}_{\hat{\Pi},\tilde{\mathcal{A}}}(\overline{x},\overline{y},z)\right\} \tag{5}$$

Naturally, we prove Eq. (5) by showing that the soundness of BZK holds in the simulation by $\hat{\mathcal{A}}$. As we have discussed in Section 4.3, the soundness of BZK holds as long as the string r sent by the verifier is at least n bits longer than the allowed length of y. This is of course true (by the definition of y). However, we must still show that the soundness is not affected when BZK is run concurrently with Π' and PRS. The key point here is that $\hat{\mathcal{A}}$ never rewinds $\tilde{\mathcal{A}}$ past a BZK message that has been sent (a fixed point is set whenever such a message is sent). Therefore, it is possible to use $\tilde{\mathcal{A}}$ in order to construct a cheating prover P^*.

Formally, let $\overline{x}$ and $\overline{y}$ be the respective input vectors of P_1 and P_2. Then, we construct a cheating prover P^* who interacts with an honest verifier V in a *single* execution of Protocol BZK. The probability that P^* successfully proves a false statement will be polynomially related to the probability that $\hat{\mathcal{A}}$ accepts the proof of a false statement from $\tilde{\mathcal{A}}$ in an execution of $\hat{\Pi}$ (when $\overline{x}$ and $\overline{y}$ are the parties' input vectors). P^* internally incorporates $\tilde{\mathcal{A}}$ and works as follows. P^* uniformly chooses one of the $\zeta p(n)$ BZK protocol executions in the $p(n)$ executions of Π and internally emulates the entire simulation of $\hat{\mathcal{A}}$ for $\tilde{\mathcal{A}}$, except with respect to the selected BZK execution. We stress that P^* emulates the weakZK-hybrid simulation by $\hat{\mathcal{A}}$ for $\tilde{\mathcal{A}}$, and does not emulate a real execution of Π or an execution of $\tilde{\Pi}$. For now, assume that this emulation can be perfectly carried out by P^* (we will show how this is done later). Now, when $\tilde{\mathcal{A}}$ reaches the selected PRS execution, P^* sends the statement being proved in this execution to V and externally forwards the messages of this execution between $\tilde{\mathcal{A}}$ and V. We stress that all other messages that are sent concurrently to this PRS execution are internally simulated by P^*. Since $\hat{\mathcal{A}}$ plays the honest verifier in its emulation, the view of $\tilde{\mathcal{A}}$ in the emulation by $\hat{\mathcal{A}}$ is identical to its view in this emulation by P^*. Noting that there are $\zeta p(n)$ proofs of PRS supplied by $\tilde{\mathcal{A}}$, we conclude that the probability that P^* successfully proves a false statement to V equals $1/\zeta p(n)$ times the probability that $\hat{\mathcal{A}}$ accepts the proof of a false statement from $\tilde{\mathcal{A}}$ in the emulation. By the (stand-alone) soundness of PRS, this probability must therefore be negligible. Eq. (2) follows.

The above analysis assumes that the emulation can be carried out perfectly by P^*. We now show that this is the case. First, the emulation of parties P_1 and $\hat{\mathcal{A}}$ can be carried out because P^* knows the inputs $\overline{x}$ and $\overline{y}$. However, P^* cannot emulate the trusted party for the weak zero-knowledge functionality. This is because the functionality cannot be efficiently computed. We solve this problem as follows. If $\tilde{\mathcal{A}}$ successfully proves a statement v, then in the emulation, P^* assumes that v is correct (and emulates the trusted party sending $(v,1)$ to P_1). If v is a correct statement, then the emulation is perfect. If v is incorrect, then the emulation may not be correct. However, up until the conclusion of this proof (where the emulation is "messed up"), the emulation was perfect. Therefore, with probability $1/\zeta p(n)$, this statement will be the one that V is verifying, and the contradiction will be derived.

STEP 3 – CONCLUSION. The proof is concluded by showing that the distribution generated by $\hat{\mathcal{A}}$ in $\hat{\Pi}$ is indistinguishable from the distribution generated by $\mathcal{A}'$ in Π'. This is identical to the proof of Eq. (3) for the case that P_1 is corrupted. (Notice that in $\hat{\Pi}$ all subproofs are proven using the ideal weak zero-knowledge functionality. Therefore, the proof is the same if it is P_1 or P_2 that is corrupted.) Thus the proof of the case that P_2 is corrupted follows by combining Equations (4), (5)

and an analog of Eq. (3).

This concludes the proof of Theorem 7. ■

Remark: We call attention to the fact that in the proof of Theorem 7, the ZK-hybrid model adversary $\mathcal{A}'$ either verifies PRS and simulates BZK (when P_1 is corrupt), or simulates PRS and verifiers BZK (when P_2 is corrupt). That is, there is never a time that $\mathcal{A}'$ has to simultaneously verify and simulate the same zero-knowledge protocol. This is a crucial point in our proof and is what enables us to ensure the soundness of the subproofs during the ZK-hybrid simulation.

5.4 Generalizing the Composition Theorem

Theorem 7 is stated so that the *same* ZK-hybrid model protocol Π' is executed concurrently. However, there is really no need to consider the same protocol. Rather, the theorem can be stated with respect to possibly different protocols for the ZK-hybrid model $\Pi'_1, \Pi'_2, \ldots$ The composition theorem will still hold as long as all the protocols replace the ideal zero-knowledge calls in the same way (as described in the proof of Theorem 7). Thus, we really demonstrate the feasibility of obtaining m-bounded concurrent composition in the real model of *arbitrary* protocols that are designed in the ZK-hybrid model.

6 Obtaining Bounded Concurrent Two-Party Computation

In this section, we show that by Theorem 7, it suffices to provide protocols that are secure in the ZK-hybrid model. We then show that such protocols exist. First, however, we formally define security in the ZK-hybrid model.

Secure computation in the ZK-hybrid model – definition. Let Π' be a two-party protocol that is designed in the ZK-hybrid model. That is, Π' contains regular interaction between the parties as well as ideal calls to the zero-knowledge functionality. Security in the ZK-hybrid model is defined in the natural way. That is,

Definition 8 (security in the ZK-hybrid model): *Let $m(\cdot)$ be a polynomial, f a functionality and Π' a protocol for the* ZK-hybrid *model. Then, we say that Π'* securely computes *f* in the ZK-hybrid model under *m*-bounded concurrent composition *if for every non-uniform polynomial-time* ZK-hybrid *adversary $\mathcal{A}'$ running Π' there exists a non-uniform polynomial-time ideal-model adversary $\mathcal{S}$ such that for every polynomial $p(n)$,*

$$\left\{\text{IDEAL}_{f,\mathcal{S}}(\overline{x},\overline{y},z)\right\}_{n\in\mathsf{N};\overline{x},\overline{y}\in(\{0,1\}^n)^{p(n)};z\in\{0,1\}^*} \stackrel{\text{c}}{\equiv} \left\{\text{ZK-HYBRID}^m_{\Pi',\mathcal{A}'}(\overline{x},\overline{y},z)\right\}_{n\in\mathsf{N};\overline{x},\overline{y}\in(\{0,1\}^n)^{p(n)};z\in\{0,1\}^*} \tag{6}$$

We will actually require that the ideal-model adversary $\mathcal{S}$ of Definition 8 is *black-box* (as defined in the paragraph following Definition 1) and carries out no rewinding of $\mathcal{A}'$. This is called straight-line or one-pass black-box simulation. Furthermore, we require that $\mathcal{S}$ "succeeds" even if $\mathcal{A}'$ runs in expected polynomial-time, where success means that Eq. (6) holds and $\mathcal{S}$ remains (strict) polynomial-time (of course, while counting each oracle call to $\mathcal{A}'$ as a single step only).[20] For the

[20] We stress that, in general, it is not guaranteed that $\mathcal{A}'$'s expected running time remains polynomial when it interacts with a simulator. This will be dealt with later on.

lack of a better term, we call an ideal-model adversary $\mathcal{S}$ who fulfills these additional requirements a strong simulator. We note that strong simulation is needed due to problems arising from composing expected polynomial-time adversaries. (If the adversary $\mathcal{A}'$ obtained from Theorem 7 ran in strict polynomial-time, then we could use *any* ideal-model adversary that fulfills Definition 8.)

Security in the ZK-hybrid model suffices. An important corollary of Theorem 7 is the fact that in order to obtain m-bounded concurrent secure two-party computation in the real model, it suffices to obtain m-bounded concurrent secure two-party computation in the ZK-hybrid model with strong simulation (see above).

Corollary 9 (bounded-concurrent secure two-party computation): *Assume that there exists a two-party protocol* Π' *that securely computes a functionality* f *in the* ZK-hybrid model *under* m*-bounded concurrent composition. Furthermore, the ideal-model adversary for* Π' *is a* strong simulator*. Then, assuming the existence of enhanced trapdoor permutations and collision resistant hash functions, there exists a two-party protocol* Π *that securely computes* f *in the* real model *under* m*-bounded concurrent composition.*

Proof: The proof of security works by showing the existence of an ideal-model adversary $\mathcal{S}$ for every real-model adversary $\mathcal{A}$. Now, let $\mathcal{A}$ be a real-model adversary for Π. Then, by Theorem 7, there exists an expected polynomial-time adversary $\mathcal{A}'$ for Π' in the ZK-hybrid model such that

$$\left\{\text{ZK-HYBRID}^{m}_{\Pi',\mathcal{A}'}(\overline{x},\overline{y},z)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}^{m}_{\Pi,\mathcal{A}}(\overline{x},\overline{y},z)\right\} \tag{7}$$

Next, by the assumption that Π' securely computes f in the ZK-hybrid model, we have that for every polynomial-time adversary $\mathcal{A}'$ there exists a polynomial-time adversary $\mathcal{S}$ for the ideal model such that

$$\left\{\text{IDEAL}_{f,\mathcal{S}}(\overline{x},\overline{y},z)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{ZK-HYBRID}^{m}_{\Pi',\mathcal{A}'}(\overline{x},\overline{y},z)\right\} \tag{8}$$

If $\mathcal{A}'$ was a (strict) polynomial-time adversary, then this would complete the proof. However, $\mathcal{A}'$ is *expected* polynomial-time and thus we have no guarantee that $\mathcal{S}$ succeeds for such an $\mathcal{A}'$. Therefore, in order to complete the proof, we utilize three facts:

1. $\mathcal{S}$ remains strict polynomial-time and "successfully" simulates even when $\mathcal{A}'$ runs in expected polynomial-time,
2. $\mathcal{S}$ uses one-pass black-box access to $\mathcal{A}'$,
3. The expected running-time of $\mathcal{A}'$ is polynomial for *every* distribution over the messages of Π' that $\mathcal{A}'$ may receive.

(The first two facts hold because $\mathcal{S}$ is strong; the third holds by Theorem 7.) First, since $\mathcal{S}$ successfully simulates even when $\mathcal{A}'$ is expected polynomial-time, we have that Eq. (8) holds. It remains to show that the combined running time of $\mathcal{S}$ and $\mathcal{A}'$ is expected polynomial-time. The fact that $\mathcal{S}$ contributes a strict polynomial number of steps to this running time is also due to the assumption that $\mathcal{S}$ is a strong simulator. The fact that $\mathcal{A}'$ contributes an expected polynomial number of steps can be seen as follows: Since $\mathcal{S}$ uses one-pass black-box access to $\mathcal{A}'$ in its simulation of Π', the simulation actually just involves $\mathcal{S}$ invoking $\mathcal{A}'$ on some distribution over the messages of Π'. Again, by the assumption on $\mathcal{A}'$, it remains expected polynomial-time for every distribution, and so also in this simulation. We conclude that for every expected polynomial-time ZK-hybrid adversary $\mathcal{A}'$

obtained via Theorem 7, there exists an expected polynomial-time ideal-model adversary $\mathcal{S}$ such that Eq. (8) holds.

By combining Equations (7) and (8) we obtain that Π securely computes f in the real model and under m-bounded concurrent composition. ■

Bounded concurrent secure computation in the plain model. By Proposition 9, all that remains is for us to demonstrate the existence of a protocol that securely computes any two-party functionality in the ZK-hybrid model and under m-bounded concurrent composition. Furthermore, there should exist a strong simulator for this protocol. However, such a protocol is already known to exist. In particular, Canetti et al. [12] show that under the assumption of the existence of enhanced trapdoor permutations, universally composable protocols exist in the ZK-hybrid model for any two-party functionality. The [12] ideal-model adversary uses one-pass black-box simulation and succeeds even when the adversary runs in expected polynomial-time (this is easily verified). Therefore, since universal composability implies m-bounded concurrent composition, we obtain the following theorem:

Theorem 10 (Theorem 1.3 restated): *Assume that enhanced trapdoor permutations and collision resistant hash functions exist. Then, for any two-party functionality f and for any polynomial $m(\cdot)$, there exists a protocol Π that securely computes f under m-bounded concurrent composition.*

Round complexity. The number of rounds in our protocol for m-bounded concurrent secure two-party computation (in the real model) is in the order of m times the number of rounds in the [12] protocol. We remark that the [12] construction (for the case of static adversaries) can be made constant-round by using the semi-honest protocol of Yao [40]. Therefore, the round complexity of our protocol is $O(m)$.

Future Work

This work leaves open a number of interesting questions. First, our protocol for the m-bounded concurrent model uses both black-box and non black-box techniques, whereas the lower bound relates only to black-box simulation. So, can a black-box secure protocol be achieved with $O(m)$ or more rounds (i.e., without using non black-box techniques)? Conversely, can a non black-box protocol be achieved for m-bounded concurrent composition that has less than m rounds? Taking a step beyond m-bounded concurrency, is it possible to achieve *unbounded* concurrent secure two-party computation? Another step beyond this model relates to a more general setting where many pairs of parties run a protocol and the adversary can corrupt any subset of them. Very little is known for this setting in the plain model. Finally, we note that we do not know how to extend our techniques for obtaining bounded concurrent secure *two-party* computation to the case of *multi-party* computation for an honest minority. This extension would also be of interest.

Acknowledgements

We express our deep thanks to Ran Canetti for all his help in this work. Among other things, some key ideas in the proof of the lower bound are due to Ran. We would also like to thank Jonathan Katz, Raphael Pass, Tal Rabin, and Alon Rosen for helpful discussions. Finally, we thank Phil MacKenzie for clarifications regarding [18].

References

[1] B. Barak. How to go beyond the black-box simulation barrier. In 42*nd FOCS*, pages 106–115, 2001.

[2] B. Barak. Constant-Round Coin-Tossing With a Man in the Middle or Realizing the Shared Random String Model. In 43*rd FOCS*, pages 345–355, 2002.

[3] B. Barak and O. Goldreich. Universal Arguments and their Applications. 17*th IEEE Conference on Computational Complexity*, pages 194–203, 2002.

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

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

[6] M. Blum. How to Prove a Theorem So No One Else Can Claim It. *Proceedings of the International Congress of Mathematicians*, Berkeley, California, USA, 1986, pp. 1444-1451.

[7] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[8] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In 42*nd FOCS*, pages 136–145, 2001. Full version available at the Cryptology ePrint Archive `http://eprint.iacr.org/`, Report #067, 2000.

[9] R. Canetti and M. Fischlin. Universally Composable Commitments. In *CRYPTO'01*, Springer-Verlag (LNCS 2139), pages 19–40, 2001.

[10] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires $\tilde{\Omega}(\log n)$ Rounds. In 33*rd STOC*, pages 570–579. 2001.

[11] R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universally Composable Two-Party Computation Without Set-Up Assumptions. In *Eurocrypt 2003,* Springer-Verlag (LNCS 2656), pages 68–86, 2003.

[12] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Computation. In 34*th STOC*, pages 494–503, 2002. Preliminary full version available from Cryptology ePrint Archive, `http://eprint.iacr.org`, Report #14, 2002.

[13] D. Chaum. Blind Signatures for Untraceable Payments. In *CRYPTO'82*, pages 199–203, 1982.

[14] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In 30*th STOC*, pages 409–418, 1998.

[15] S. Even, O. Goldreich and A. Lempel. A randomized protocol for signing contracts. In *Communications of the ACM,* 28(6):637–647, 1985.

[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, D. Lapidot and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Under General Assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.

[18] J. Garay and P. Mackenzie. Concurrent Oblivious Transfer. In 41*st FOCS*, pages 314–324, 2000.

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

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

[21] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* To be published. Available from `http://www.wisdom.weizmann.ac.il/~oded/foc-vol2.html`.

[22] O. Goldreich, S. Goldwasser and S. Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):792–807, 1986.

[23] O. Goldreich and A. Kahan. How To Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.

[24] O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. In *CRYPTO 2001*, Springer-Verlag (LNCS 2139), pages 408–432, 2001.

[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 [19].

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

[27] S. Goldwasser, S. Micali, and R.L. Rivest. A digital signature scheme secure against adaptive chosen- message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[28] R. Impagliazzo and M. Luby. One-way Functions are Essential for Complexity Based Cryptography. In 30*th FOCS*, pages 230–235, 1989.

[29] J. Kilian and E. Petrank. Concurrent and resettable zero-knowledge in poly-logartihmic rounds. In 33*rd STOC,* pages 560–569, 2001.

[30] J. Kilian, E. Petrank and C. Rackoff. Lower Bounds for Zero Knowledge on the Internet. In 39*th FOCS,* pages 484–492, 1998.

[31] L. Lamport. Constructing Digital Signatures from One-Way Functions. *SRI International,* CSL-98, 1979.

[32] Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. To appear in the *Journal of Cryptology,* 2003.

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

[34] R. Pass. Simulation in Quasi-Polynomial Time, and Its Application to Protocol Composition. In *Eurocrypt 2003*, Springer-Verlag (LNCS 2656), pages 160–176, 2003.

[35] M. Prabhakaran, A. Rosen and A. Sahai. Concurrent Zero Knowledge With Logarithmic Round Complexity. In 43*rd FOCS,* pages 366–375, 2002.

[36] M. Rabin. How to exchange secrets by oblivious transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.

[37] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *Eurocrypt'99*, Springer-Verlag (LNCS 1592), pages 415–413, 1999.

[38] J. Rompel. One-Way Functions are Necessary and Sufficient for Secure Signatures. In 22*nd STOC*, pages 387–394, 1990.

[39] A. Rosen. A Note on the Round-Complexity of Concurrent Zero-Knowledge. In *CRYPTO 2000,* Springer-Verlag (LNCS 1880), pages 451–468, 2000.

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

A Blum's Protocol for Hamiltonicity [6]

The zero-knowledge proof of [35] uses n parallel repetitions of the following basic proof system for the NP-complete *Hamiltonian Cycle* (HC) problem. We consider directed graphs (and the existence of directed Hamiltonian cycles).

Construction 2 (Basic proof system for HC):

- Common Input: *a directed graph* $G = (V, E)$ *with* $n \stackrel{\text{def}}{=} |V|$.
- Auxiliary Input to Prover: *a directed Hamiltonian Cycle,* $C \subset E$*, in* G.
- Prover's first step (P1): *The prover selects a random permutation,* π*, of the vertices* V*, and commits (using a perfectly-binding commitment scheme) to the entries of the adjacency matrix of the resulting permuted graph. That is, it sends an* n*-by-*n *matrix of commitments so that the* $(\pi(i), \pi(j))^{\text{th}}$ *entry is a commitment to 1 if* $(i, j) \in E$*, and is a commitment to 0 otherwise.*
- Verifier's first step (V1): *The verifier uniformly selects* $\sigma \in \{0, 1\}$ *and sends it to the prover.*
- Prover's second step (P2): *If* $\sigma = 0$ *then the prover sends* π *to the verifier along with the revealing (i.e., preimages) of all commitments. Otherwise, the prover reveals to the verifier only the commitments to entries* $(\pi(i), \pi(j))$ *with* $(i, j) \in C$*. In both cases the prover also supplies the corresponding decommitments.*
- Verifier's second step (V2): *If* $\sigma = 0$ *then the verifier checks that the revealed graph is indeed isomorphic, via* π*, to* G*. Otherwise, the verifier just checks that all revealed values are 1 and that the corresponding entries form a simple* n*-cycle. In both cases the verifier checks that the decommitments are proper (i.e., that they fits the corresponding commitments). The verifier accepts if and only if the corresponding condition holds.*

Proposition 11 *The protocol which results by* n *parallel repetitions of Construction* 2 *is an interactive proof of Hamiltonicity with soundness error* 2^{-n}.

An important property of Construction 2 is that it is possible to simulate if given the n-bit verifier-query string $q = q_1, \ldots, q_n$ before the prover's P1 message is sent. Specifically, in the i^{th} repetition, if $q_i = 0$ then the simulator generates a commitment to a random permutation of the input graph, and if $q_i = 1$ then it generates a commitment to a graph that contains a random simple n-cycle only. Clearly, the simulator can then answer the verifier queries.