Protocols for Bounded-Concurrent Secure Two-Party Computation in the Plain Model*

Yehuda Lindell

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

May 18, 2004

Abstract

Until recently, most research on the topic of secure computation focused on the stand-alone model, where a single protocol execution takes place. In this paper, we construct protocols for the setting of *bounded-concurrent self composition*, where a (single) secure protocol is run many times concurrently, and there is a predetermined bound on the number of concurrent executions. In short, we show that *any* two-party functionality can be securely computed under bounded-concurrent self composition, in the plain model (where the only setup assumption made is that the parties communicate via authenticated channels). Our protocol provides the first feasibility result for general two-party computation in the plain model, *for any model of concurrency*. All previous protocols assumed a trusted setup phase in order to obtain a common reference string. On the downside, the number of rounds of communication in our protocol is super-linear in the bound on the number of concurrent executions. However, we believe that our constructions will lead to more efficient protocols for this task.

Keywords: secure two-party computation, concurrent self composition, setup assumptions.

*The results in this paper appeared in an extended abstract in [27].

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 [39, 22].

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 [25, 38, 11, 24, 36]. Other specific problems that have been studied in the context of concurrent composition are oblivious transfer [17] and authenticated Byzantine agreement [31]. In all of the above-mentioned work, the setting that is considered is that of concurrent self composition, where a *single* protocol is run concurrently any polynomial number of times in a network (we stress, that the secure protocol in question is the only protocol being executed). Furthermore, all the executions are run by the same set of parties, and each party plays the same role throughout all of these executions (e.g., for zero knowledge, one party always plays the prover and the other party always plays the verifier).[1]

A different notion, called general composition, considers a very broad setting where many sets of (possibly different) parties run many protocols, and secure protocols may run concurrently with arbitrary other protocols.[2] Such a setting realistically models the security needs of modern networks

[1] Actually, such a setting should be called concurrent self composition with a single set of parties. See [29] for a full taxonomy of types of protocol composition.

[2] According to the taxonomy of [29], this should actually be called concurrent general composition with arbitrary sets of parties.

like the Internet, where many different arbitrary protocols are executed by many different sets of parties (with possibly related inputs). The notion of concurrent general composition was first studied by [35] who considered the case that a secure protocol is executed *once* concurrently with another arbitrary protocol. The unbounded case, where a secure protocol can be run any polynomial number of times in an arbitrary network, was then considered by the framework of universal composability [8]. Loosely speaking, universal composability is a specific security definition with the important property that any protocol that is proven secure under this definition is guaranteed to remain secure under (unbounded) concurrent general composition. Due to the robust security guarantees that are provided, universally composable protocols are highly desirable. Fortunately, such protocols exist. In fact, it has been shown that assuming an honest majority, *any* multi-party functionality can be securely computed in a universally composable way [8]. Furthermore, in the *common reference string model*[3], any two-party or multi-party functionality can be computed under this definition, for *any* number of corrupted parties.

On the negative side, in the plain model and without an honest majority (as in the two-party case), there exist large classes of functionalities that cannot be securely computed according to the definitions of universal composability [10, 8, 12]. Note that these impossibility results relate specifically to the definition of universal composability and not necessarily to the possibility of obtaining protocols that are secure under concurrent general composition by some other definition. Nevertheless, impossibility results have also been demonstrated for any definition achieving concurrent *general* composition [28], and even for concurrent *self* composition [30], where impossibility holds in the plain model and in the case of no honest majority. (We note that the classes of functionalities that cannot be securely computed are not exactly the same for all these impossibility results. However, large classes of functionalities are ruled out in each case.)

Bounded versus unbounded concurrency. As we have described above, broad impossibility results exist for both concurrent general and self composition. However, the impossibility results in [30] for concurrent self composition rely inherently on the fact that the secure protocol in question can be run concurrently *any* polynomial number times; this property is called unbounded concurrency. In contrast, a more limited model of concurrency, first considered by [1], restricts the number of concurrent executions to be some fixed, predetermined polynomial in the security parameter. Furthermore, the protocol design can depend on this number (i.e., if the bound is n^2, then the protocol can be designed so that it is secure for at most n^2 concurrent executions, and no more). This model is called bounded concurrency, and when m is the maximum number of concurrent executions, we talk about m-bounded concurrent composition (note that $m = m(n)$ is a fixed polynomial in the security parameter). Importantly, the possibility of obtaining protocols that are secure under m-bounded concurrent self composition was not ruled out in [30].

The model of concurrency. Before proceeding to describe our results, we describe the model of concurrency considered here in more detail. In this work, we consider self composition of two-party protocols, where a single pair of parties run the same protocol many times concurrently. 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

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

model some real-world scenarios, like a number of servers concurrently interacting with many clients (and where we assume that there is no simultaneous corruption of a server and a client). As we have mentioned above, this is the exact model considered in the entire body of work on concurrent zero-knowledge. Note that this is a rather limited model, both because self composition (rather than general composition) is considered, and because only a single set of parties is considered (equivalently, the adversary is severely limited in its corruption capability). However, we stress that no general feasibility results have been shown for *any* model of concurrency, without assuming some trusted setup phase (beyond that required for authenticated channels) or an honest majority. This therefore serves as a good first step. In addition, studying more limited models deepens our understanding of exactly where the line between feasibility and infeasibility lies, thus providing an "explanation" of sorts as to the source of the impossibility results for concurrent composition. (We note that the fact that we consider self, rather than general, composition is also very limiting. However, this is essential due to the impossibility results of [28] that hold even when the secure protocol is executed only once in the setting of concurrent general composition.)

Our results. In this work, we show that any two-party functionality can be securely computed under bounded-concurrent self composition and in the plain model. More specifically, we prove the following theorem:

Theorem 1 *Assume that enhanced trapdoor permutations*[4] *and collision resistant hash functions exist. Then, for any probabilistic polynomial-time two-party functionality f and for any m, there exists a protocol Π that securely computes f under m-bounded concurrent composition in the plain model.*

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 [13]. 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 statements with

[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 [19, Appendix C]. We note that the construction of [22] for secure two-party computation in the stand-alone model also 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.

the black-box zero-knowledge protocol of [37], while the other party proves statements with 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 [37] is more problematic. We therefore devise a *new* black-box simulation strategy for [37] 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. The number of rounds of communication in our protocol is $O(m \cdot \kappa(n))$, where n is the security parameter and $\kappa(\cdot)$ is any super-constant function (e.g., $\kappa(n) = \log \log n$). We note that a lower bound of m rounds for protocols that are proven using only black-box simulation has been proven in [27]. Nevertheless, our protocol is proven using both black-box and non black-box techniques (and is thus not a "tight" upper-bound).

We conclude with a remark about *efficiency*. Our protocol requires $O(m\kappa(n))$ 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 any reasonable model of concurrency is of great importance, irrespective of efficiency. Furthermore, our constructions may lead to more efficient protocols (see below in subsequent work).

Discussion: setup assumptions and authenticated channels. The plain model classically refers to a scenario where the honest parties communicate via authenticated channels, but no additional setup is assumed. In most works, the assumption on the existence of authenticated channels is not articulated as a "setup assumption", but rather as part of the network model. However, in reality, some sort of setup assumption is also required for achieving authenticated channels. We argue that this assumption should be explicitly treated as such.

We note that despite the fact that authenticated channels *is* a setup assumption, it is a strictly weaker one than that required for obtaining a common reference string, for example. Specifically, in order to achieve authenticated channels, it suffices to have a certificate authority who reliably posts keys on a bulletin board [9]. (We stress that there is no need for the certificate authority to run any protocol with the parties posting keys and, in particular, proofs of knowledge are not necessary.) Furthermore, upon inspection of the lower bounds for concurrent self composition of [30], it is easy to see that they extend to a model where such a "secure bulletin board" exists. Thus, a common reference string cannot be generated from the same assumption used for authenticated channels (or else the lower bounds of [30] would not hold).

Subsequent and related work. Subsequent to this work, a *constant-round* protocol for bounded-concurrent secure two-party computation was presented in [34]. This result was later improved and extended in [33] to provide a protocol that **(a)** works for multiparty functionalities, and **(b)** can be extended to the general setting where arbitrary sets of parties run the protocol (and any subset of parties may be corrupted). Both of the above works build on our protocol and take our exact framework, while replacing the specific zero-knowledge protocols with alternative ones. Thus, our constructions have already served as the basis for significant improvements and extensions.

We note that both our protocol and the protocols of [34] and [33] suffer from very high communication complexity in bandwidth. Specifically, for m concurrent executions, the bandwidth of the protocols are at least $\Omega(mn^2)$, where n is the security parameter. We note that a linear dependence of the bandwidth on m is actually inherent. That is, it has been shown that there exist many functionalities such that any protocol that securely computes these functionalities under m-bounded concurrent self composition must have communication complexity of at least m [30].

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 [23, 32, 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. All parties and the adversary run in time that is polynomial in 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. (According to this description, all the honest party's inputs are fixed at the onset. However, our protocols are also secure when the honest party's inputs may be chosen adaptively throughout the execution. This will be discussed later.) 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 behaviour. 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.[6] The focus of this work is not on fairness. We therefore present a definition where the adversary always receives its own output

[6] We follow the approach of [19] by defining the two-party case as one where the adversary controls exactly one of the parties. This is in contrast to a more general approach in which the adversary may control zero, one or both parties. This approach somewhat simplifies the presentation, without weakening the results. Specifically, assuming *authenticated channels*, any protocol that is secure when exactly one party is corrupted can be transformed into a protocol that is secure when any number (i.e., 0, 1 or 2) of the parties are corrupted, in the following way. First, using the given authenticated channels, the two parties establish secure channels for private and reliable communication. Next, the original protocol is executed over this private channel. It follows that if no parties are corrupted, then the adversary learns nothing (except for the length of the messages and so the length of the inputs). Furthermore, if one party is corrupted, then the effect is exactly the same as for the original protocol, where security is guaranteed. Finally, the case of both parties corrupted is always straightforward. We conclude that the resulting protocol is secure in the more general case, as required.

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 many 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 each 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.[7]

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 ith 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 polynomial-time machine (representing the ideal-model adversary).

[7] The fact that the adversary can query the trusted party once for each execution is essential for obtaining meaningful security. Otherwise, the adversary (controlling P_2) could obtain a series of values $f_2(x,y), f_2(x,y'), f_2(x,y''), \ldots$ for a single input value x belonging to the honest party. Furthermore, it may choose y, y', y'' etc. adaptively based on previous outputs. This may reveal much more information than intended. For example, if f is the "less than" function, then the adversary could conduct a binary search on the input range and find the exact value of x.

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.

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 ith input into the ith 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 ith execution of Π and replies according to the instructions of Π and this view.[8] 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 ith 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 polynomial-time*

[8] 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 $p(n)$,

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

Remark – adaptively chosen inputs. Definition 1 is overly simplistic and limited in that it assumes that the honest party's inputs are fixed before any execution begins. However, our protocol also works for a more general definition where inputs for later executions can depend on outputs of earlier executions; see [30] for such a definition. We present this simpler definition for the sake of clarity and because the issue of adaptively chosen inputs versus a-priori fixed inputs actually has no effect on our construction.

3 Tools for our Protocol – Zero-Knowledge

In this section, we develop the basic tools needed for proving Theorem 1. As we have described in the Introduction, this involves the construction of zero-knowledge proofs that can be used as subprotocols within any other protocol, in the setting of bounded concurrency. We use the standard definitions for zero-knowledge and proofs of knowledge, see [18], and assume familiarity with these notions.

3.1 The Zero-Knowledge Arguments 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.[9] 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. The remainder of this section describes how this is achieved.

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. As we will see below, the extractor for the commit-with-extract scheme of [4] works by simulating a zero-knowledge proof of membership. We begin by describing how such a commitment 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 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 provided by the prover in the second

[9] In a *proof of membership*, the verifier is convinced that a certain statement is correct. However, in a *proof of knowledge*, the verifier is also convinced that the prover actually holds a witness for this statement.

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 1 (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.*[10]

We note that [4] also demonstrate the existence of a witness-extended emulator for Protocol 1, as defined by [26]. Loosely speaking, a witness-extended emulator is a machine that not only outputs a witness with the required probability (as is required from a knowledge extractor), but also outputs a simulated transcript of the interaction between the prover and the verifier. Furthermore, whenever the transcript of the interaction is such that the verifier accepts, then the witness that is obtained is valid. To explain this further, consider a case that the prover convinces the verifier in a real interaction with probability p. Then, with probability negligibly close to p, the above-described machine should output an accepting transcript and a valid witness. Furthermore, with probability negligibly close to $1-p$, the machine should output a rejecting transcript (and we don't care about the witness). We note that the transcript that is obtained is also *publicly verifiable*, meaning that the verifier's accept or reject decision can be efficiently obtained from the transcript. Witness-extended emulation is often needed when a proof of knowledge is used as a *subprotocol* within a larger protocol; as is the case for our constructions. See [4] for more discussion on witness-extended emulation.

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 $\langle y, v\rangle$ to the receiver, where $v = b(r) \oplus \sigma$. 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. By the security of the coin-tossing protocol, y is pseudorandom. Therefore, the hiding property remains essentially the same as in the original scheme. Likewise, because f is a permutation, any y defines a unique value $b(f^{-1}(y)) = b(r)$ and thus the scheme remains perfectly binding (i.e., given any pair (y, v), the committed value is uniquely defined by $v \oplus b(f^{-1}(y))$). 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 2 (commit-with-extract bit commitment scheme):

- **Input:** *The committer has a bit σ to be committed to.*

[10] 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 $(x, w) \in R$.

- **Commit phase:**
  1. The committer chooses an enhanced trapdoor permutation:[11]

     The committer chooses an enhanced trapdoor permutation f along with its trapdoor t and sends f to the receiver.[12]
  2. The parties run a coin-tossing protocol:
     (a) *The receiver chooses a random string $r_1 \in_R \{0,1\}^n$ and sends $c = \mathsf{Commit}(r_1; s)$ to the committer (using any perfectly-binding commitment scheme* Commit *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 $c = \mathsf{Commit}(r_1; s)$.*
     (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 how a commitment extractor CK can obtain the value σ committed to by the committer. 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 the receiver 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 honestly 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

[11] See [19, 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 $\{0,1\}^n$.

[12] For the sake of simplicity, we assume that the protocol uses *certified* [15] 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].

strategy relies only on the ability to simulate the zero-knowledge proof of membership in step 2d. See [4] for a formal definition of commit-with-extract schemes and for a full proof of Protocol 2.

Summary. Consider the system of zero-knowledge arguments of knowledge of Protocol 1 where the commit-with-extract scheme that is used is that of Protocol 2. Then, it holds that the *simulation* of the combined protocol works by simulating the zero-knowledge proof of membership of Part 2 that is provided by P. Furthermore, the *extraction* strategy of the combined protocol works by simulating the zero-knowledge proof of membership of Part 1 that is provided by V. Actually, both the simulation and extraction strategies also involve other instructions, as described above. However, these instructions do not require any "rewinding" or involved simulation strategies; rather they can be generated using one-pass black-box simulation (specifically, these instructions can be generated independently of the internal simulation of the subproofs and based solely on the transcript of the other messages). Due to this fact, we have that if the zero-knowledge subproofs can be simulated under *concurrent composition* (even with arbitrary other protocols), then both simulation and extraction of Protocol 1 can be carried out under *concurrent composition* (even with arbitrary other protocols). This important property is summarized in the following informal lemma:

Lemma 3.1 (informal lemma): *Let (P, V) be the protocol obtained by plugging the commit-with-extract scheme of Protocol 2 into Protocol 1. Furthermore, assume that the subproofs of membership (in Protocol 2 and in part 2 of Protocol 1) constitute zero-knowledge proof or argument systems with negligible soundness error, even when run concurrently with each other and arbitrary other protocols. Then,*

1. *(P, V) is a zero-knowledge argument system, even when run concurrently with itself and arbitrary other protocols.*
2. *(P, V) is an argument of knowledge with negligible knowledge error, even when run concurrently with itself and arbitrary other protocols. Furthermore, there exists a witness-extended emulator for (P, V) in this setting.*

The proof of this lemma follows directly from the proofs that Protocol 2 is a commit-with-extract commitment scheme and that Protocol 1 is a system of zero-knowledge proofs of knowledge, as shown in [4]. A formal version of Lemma 3.1 is stated and proved in Appendix A.

3.2 Black-Box Bounded Concurrent ZK

In this section we describe the concurrent zero-knowledge protocol of Richardson and Kilian [37], with a certain setting of parameters relating to the number of rounds. It has been shown that the protocol of [37] with a poly-logarithmic number of rounds is (unbounded) concurrent zero-knowledge [24]. In contrast, we extend the protocol to have $O(m{\cdot}\kappa(n))$ rounds, where m is the maximum number of concurrent executions and $\kappa(\cdot)$ is any super-constant function; i.e., $\kappa(n) = \omega(1)$.[13] We then show that this extended protocol is secure under m-bounded concurrent composition. Although this is a far weaker claim (i.e., requiring more rounds and achieving only bounded concurrency), the simulation strategies of [37, 24] are problematic when the protocol is used as a module within another larger protocol (as will be the case here). We therefore increase the number of rounds and

[13] Actually, we will assume that $\kappa(n)$ is at most polynomial in n; therefore, it cannot be *any* super-constant function.

provide a *new* simulation strategy that also succeeds when the protocol is plugged into another (larger) protocol.

Before presenting the actual protocol of [37], we describe it on an informal level. The idea behind the protocol is to provide the simulator with many opportunities to rewind the verifier. This is achieved by having a preamble of many iterations, where rewinding in just one of the iterations suffices for simulating the entire proof. To demonstrate this idea, we consider the case that the preamble consists of just one iteration. Then, the protocol begins with the verifier sending a perfectly-hiding commitment to a random string $r \in_R \{0,1\}^n$. Next, the prover sends a perfectly-binding commitment $c = \mathsf{Commit}(0^n)$, after which the verifier decommits to r. (Note that $\mathsf{Commit}(0^n)$ is actually a randomized computation.) Finally, the prover proves a zero-knowledge (or witness-indistinguishable [16]) proof to the verifier that either the statement in question is true, or that its commitment c equals $\mathsf{Commit}(r)$. Now, the honest prover will always prove that the statement in question is true. Furthermore, due to the fact that the verifier's commitment is perfectly-hiding, a cheating prover has no information on r when it sends c to the verifier. Therefore, except with negligible probability, it will not be the case that $c = \mathsf{Commit}(r)$. Thus, if the verifier is convinced, the statement in question must be true (by reduction to the soundness of the zero-knowledge or witness-indistinguishable proof provided at the end). The fact that the proof is zero-knowledge can be seen as follows: The simulator is able to rewind the verifier after obtaining the decommitment value r. Then, it can continue the proof by providing a commitment c such that indeed $c = \mathsf{Commit}(r)$. In this case, the simulator can prove the proof at the end using the fact that its commitment is to r and without having any witness to the fact that the statement in question is true. In the actual proof system, this preamble is repeated many times, presenting many opportunities for rewinding the verifier. We now present the proof system:

Protocol 3 (Protocol RKZK – concurrent ZK proof for a language L):

- **Common input:** *$x \in L$ and a security parameter n.*
- **Auxiliary input to the prover:** *a witness w for $x \in L$.*
- **Preamble length:** ℓ
- **Part 1 – the preamble:**
  1. *P sends V the first message of a two-round perfectly hiding commitment scheme.*
  2. *V chooses ℓ random strings $r_1, \ldots, r_\ell \in_R \{0,1\}^n$. Then, V sends P perfectly-hiding commitments to $r_1, \ldots, r_\ell$.*
  3. *Repeat ℓ times (for $j = 1, \ldots, \ell$):*
     - *(a) P computes a perfectly-binding commitment $c_j = \mathsf{Commit}(0^n)$ and sends it to V.*
     - *(b) V decommits, revealing r_j.*
- **Part 2 – the proof:**
  1. *P and V run any (efficient-prover) zero-knowledge or witness-indistinguishable proof for $\mathcal{NP}$ for the following statement:*

     Either $x \in L$ or there exists a j $(1 \leq j \leq \ell)$ such that $c_j = \mathsf{Commit}(r_j)$.

Let $\mathsf{RKZK}(\ell)$ denote the instantiation of Protocol RKZK for the case that the number of iterations in the preamble is ℓ. Now, as we have mentioned above, it has been shown that $\mathsf{RKZK}(\tilde{O}(\log^2 n))$ is unbounded concurrent zero-knowledge [24]. Below, we will prove that $\mathsf{RKZK}(m\kappa(n))$ is m-bounded

concurrent zero-knowledge. This is a far weaker claim than that of [24]. However, as we have mentioned, we prove this in order to demonstrate our new simulation technique, that enables us to later use RKZK as a subprotocol in another, different protocol. We remark that our proof that $\mathsf{RKZK}(m\kappa(n))$ is m-bounded concurrent zero-knowledge is far simpler than all known proofs for the unbounded 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.)

Proposition 2 *Let $\kappa(\cdot)$ be any super-constant function. Then, assuming the security of the prescribed commitment schemes, Protocol $\mathsf{RKZK}(m\kappa(n))$ is an m-bounded (black-box) concurrent zero-knowledge proof system for the language L. Furthermore, the zero-knowledge simulator runs in strict polynomial-time.*

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 zero-knowledge proof of part 2. We proceed to prove soundness:

Soundness: By the soundness of the proof in part 2, it suffices to show that the probability that there exists a j such that $c_j = \mathsf{Commit}(r_j)$ is negligible. However, this follows immediately from the fact that V commits using a perfectly-hiding commitment scheme. Therefore, P has no information about r_j when it generates its commitment c_j.

***m*-bounded concurrent zero-knowledge:** We begin by presenting some notation and terminology. Denote by Π_k the kth execution of the protocol and by Π_k^j the jth iteration of the preamble in the kth execution. We say that iteration Π_k^j opens when the prover P sends the perfectly-binding commitment c_j in Π_k, and that it closes when the verifier V^* decommits to r_j in Π_k. An execution is said to have been $\mathsf{satisfied}$ by the simulator S if V aborts the execution (by say, refusing to decommit in some iteration), or if there exists a j such that $c_j = \mathsf{Commit}(r_j)$. 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 $c_j = \mathsf{Commit}(r_j)$, the simulator can use this fact instead of a witness for $x \in L$ in part 2 of the proof.) Therefore, the simulation strategy is basically for S to satisfy every execution before it reaches part 2 of the proof.

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_k^j, and let r_j be the string decommitted to by the verifier V^*. Then, S rewinds V^* to the point in execution k that iteration j opened and sends a new perfectly-binding commitment $c_j = \mathsf{Commit}(r_j)$. Next, S continues forward (playing the honest prover) hoping that once again Π_k^j 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 V^* now decommits to a different string $r'_j \neq r_j$. However, due to the computational binding of the commitment scheme used by the verifier, this happens with only negligible probability.) If Π_k^j is not the first iteration to close, then S repeats the process again for a fixed polynomial number of times. If success is not obtained in any of these rewinding attempts, then S abandons this attempt to satisfy the execution and continues in the same manner as above. This strategy is used for all executions, and we show that eventually, all executions are satisfied except with negligible probability. This high-level strategy is what also lies behind the simulators of [37, 24]. The novelty of our strategy is that the simulator will iteratively fix the transcript throughout the simulation, thereby limiting the rewinding (to the part not yet fixed). Thus, we obtain a strategy which uses rewinding, but in a significantly more restricted way than for previous simulators. This will be crucial for using the zero-knowledge proof as a subprotocol. 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 verifier $V^*(x, z, r)$ in a black-box manner, playing the honest prover strategy for part 1 of the proof and waiting for one of the following two events:

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

2. *A potential iteration closes:* Let this iteration be Π_k^j and let r_j be the string decommitted to by the verifier when Π_k^j closed. Furthermore, let r_S be the random coins used by S in the simulation since the last fixed point was set. Now, S rewinds V^* back to the point at which iteration Π_k^j opened and sends $c_j = \mathsf{Commit}(r_j)$, instead of a commitment to 0^n. Then, S continues in the simulation with the hope that Π_k^j will be the first iteration to close again. We stress that S uses fresh random coins in this continuation (i.e., coins that are independent from the coins used previously). Now, as we have mentioned, S's aim in this rewinding is to once again have Π_k^j be the first potential iteration to close. Therefore, if V^* aborts an execution or closes a different potential iteration *before* Π_k^j closes, then S rewinds back to the point at which Π_k^j opened and tries again (with new random coins). S attempts this rewinding $2mn^2$ times. If in all of these attempts Π_k^j does not close before another event causing a fixed point occurs, then S abandons this attempt at satisfying Π_k. Instead, S rewinds V^* to the current fixed point and replays the honest prover strategy using the random coins r_S that it used the first time that Π_k^j closed. Then, S fixes the transcript at the point that Π_k^j closes and continues as above. In such a case, we say that the fixed point set by S is a failed fixed point. If at any point during the simulation it turns out that $\kappa(n)$ *consecutive* fixed points are failed, then S outputs failure and halts.

   Now, if in one of the $2mn^2$ rewinding attempts, Π_k^j *is* the first iteration to close (without the other events occurring), then S does the following. Let r'_j be the decommitment of V^* in the second time that Π_k^j is the first iteration to close. If $r'_j \neq r_j$, then S outputs failure and halts. Otherwise, the execution is satisfied (because $c_j = \mathsf{Commit}(r_j)$. Having satisfied Π_k, simulator 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.

In addition to the above, whenever part 2 of a satisfied (and non-aborted) execution is reached, S proves the proof using the witness to the fact that $c_j = \mathsf{Commit}(r_j)$ for some j. In contrast, if part 2 of an *unsatisfied* execution is reached, then S outputs a special unsatisfied symbol and halts.

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

(In such a case $c_j \neq \mathsf{Commit}(r_j)$ for every j, and so S cannot prove the proof of part 2.) S continues in this way until V^* concludes the interaction. This completes the description of S.

Proof of the simulation strategy: We first comment that the simulator runs in time that is polynomial in n. This follows from the fact that the overall number of rewinding attempts in all of the $p(n) \cdot m\kappa(n)$ iterations (over all executions) is at most $2mn^2 \cdot p(n) \cdot m\kappa(n) = \text{poly}(n)$. (Recall that $m = m(n)$ equals the number of executions that are run simultaneously and $p(n)$ equals the total number of executions.)

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 failure or unsatisfied symbol.

Claim 3.2 *The probability that S outputs* failure *is negligible.*

Proof: There are two events that can cause S to output failure. We separately show that each of these events can occur with at most negligible probability:

1. *Event 1 – V^* decommits to some $r'_j \neq r_j$ in one of the rewinding attempts:* The fact that V^* decommits to the same r_j before and after rewinding is due to the computational binding of the commitment scheme. The proof of this is standard and is therefore omitted.

2. *Event 2 – $\kappa(n)$ consecutive fixed points are failed:* In order to bound the probability that this occurs, we first consider the probability that a single fixed point is failed. Let Π_k^j be a potential iteration that closed first, and let p_k^j be the probability that given the transcript until the point that Π_k^j opens, iteration Π_k^j closes before any other event causing a fixed point occurs. Then, there are two possibilities:

   (a) $p_k^j \geq \frac{1}{mn}$: In this case, the probability that Π_k^j will *not* be the first to close in all of the $2mn^2$ rewinding attempts is negligible. This can be seen as follows. Denote by $\tilde{p}_k^j$, the probability that given the transcript until Π_k^j opened, iteration Π_k^j is the first iteration to close during a rewinding attempt. Now, p_k^j and $\tilde{p}_k^j$ may differ because the perfectly-binding commitment provided by S before rewinding is to 0^n and after rewinding is to r_j. However, by the computational hiding of the commitment, this difference can be at most negligible. (The formal reduction to breaking the hiding of the scheme is standard and is therefore omitted.) It therefore follows that $\tilde{p}_k^j > \frac{1}{2mn}$. In other words, the probability that Π_k^j is not the first to close in any given rewinding attempt is less than $(1 - \frac{1}{2mn})$. Therefore, the probability that it is not the first to close in all of the $2mn^2$ independent rewinding attempts is upper-bound by $(1 - \frac{1}{2mn})^{2mn^2} < e^{-n}$, which is negligible.

   (b) $p_k^j < \frac{1}{mn}$: In this case, we cannot claim that with overwhelming probability, Π_k^j will be the first iteration to close again in at least one of the rewinding attempts. Rather, we bound the probability that a potential iteration Π_k^j with $p_k^j < 1/mn$ will be the first iteration to close *before* any rewinding takes place. Let F denote the set of *potential* iterations for which the probability that the iteration is the first to close is less than $1/mn$. Then, since there are at most m potential iterations between any two fixed points (at most one per execution), we have that $|F| \leq m$. (The fact that every unsatisfied execution can only have one potential iteration between any two fixed points follows from the fact that as soon as a potential iteration closes, a fixed point is set. Thus,

a second potential iteration can only open after the previous one closed, meaning that a fixed point separates them.) Therefore, by the union bound, the probability that an iteration from F will be the first to close is less than $1/n$.

Combining the above, we have that the probability of encountering any single failed fixed point is less than $1/n + e^{-n} < 2/n$. Now, by the simulation strategy, S outputs failure if $\kappa(n)$ consecutive fixed points are failed. The probability that this occurs can be bound as follows. For any given starting fixed point, the probability that $\kappa(n)$ consecutive failed fixed points are encountered from this starting point is less than $(\frac{2}{n})^{\kappa(n)}$ (this is based on the above analysis that bounds the probability of encountering a single failed fixed point, given the previously fixed transcript, by $2/n$). There are at most $m\kappa(n)$ starting fixed points, and so the overall probability of encountering $\kappa(n)$ consecutive failed fixed points is upper bound by $m\kappa(n)(\frac{2}{n})^{\kappa(n)}$, which is negligible as long as $\kappa(n)$ is a super-constant function of n (e.g., $\kappa(n) = \log\log n$).

This completes the proof. ■

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

Claim 3.3 *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; let Π_k be this execution. We can assume that S does not output failure (because if it did, then it would not output unsatisfied). This implies that there are at most $\kappa(n)-1$ consecutive failed fixed points and that for all other fixed points, the execution for which the fixed point was set is satisfied. Now, there are $m\kappa(n)$ iterations in Π_k's preamble. Furthermore, as we have mentioned, at most one iteration of any unsatisfied execution, including Π_k, can be opened between every two fixed points. Therefore, S must have placed $m\kappa(n)$ fixed points before part 2 of Π_k is reached. Since there are at most $\kappa(n) - 1$ consecutive failed fixed points, by a pigeon-hole argument we have that there must be m *non-failed* fixed points that were placed by S. However, a non-failed fixed point is only placed when S satisfies an execution. Therefore, m different executions were satisfied between the time that Π_k started and finished. Since the setting here is of m-bounded concurrency, there can be at most $m-1$ other executions that run simultaneously with Π_k. We therefore conclude that Π_k must also 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 3.4 *Let $m = m(n)$ be a polynomial. Then, for every non-uniform probabilistic polynomial-time verifier V^* who schedules the messages according to m-bounded concurrency and every polynomial $p(\cdot)$,*

$$\left\{\langle P, V^*(z,r)\rangle(\overline{x})\right\}_{n\in\mathsf{N};\overline{x}\in L_n^{p(n)};z,r\in\{0,1\}^*} \stackrel{\mathrm{c}}{\equiv} \left\{S^{V^*(\overline{x},z,r)}(\overline{x})\right\}_{n\in\mathsf{N};\overline{x}\in L_n^{p(n)};z,r\in\{0,1\}^*}$$

where $L_n = L \cap \{0,1\}^n$.

Proof: We assume that S does not output failure or unsatisfied. This is fine because these events happen with at most negligible probability. Next, consider the following hybrid experiment. Let M^{V^*} be a machine who is given a witness to the fact that $x \in L$ and 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 the fact that $c_j = \mathsf{Commit}(r_j)$ for some j.

We first claim that the distribution generated by M^{V^*} is computationally indistinguishable from the distribution generated in a real execution between P and V^*. Intuitively, the only difference between the transcripts is regarding the prover commitments, which are to 0^n in a real execution and to r_j in the simulation by M^{V^*}. Therefore, the indistinguishability is due to the hiding property of commitments. More formally, consider the following two experiments with a probabilistic polynomial-time distinguishing machine D and a perfectly-binding commitment scheme Commit. In the first experiment, D adaptively chooses n-bit values $r_1, r_2, \ldots$ and sends these values to an oracle that returns $c_j = \mathsf{Commit}(r_j)$ for each j. Note that D chooses the r_j values adaptively and therefore r_j is chosen after $c_1, \ldots, c_{j-1}$ have been received. In the second experiment, D does the same; however, for each oracle query it receives back a commitment $c_j = \mathsf{Commit}(0^n)$. (Of course, all commitments are generated with independent random coins.) By the hiding property of the commitments, these two experiments are indistinguishable; i.e., every such D outputs 1 in both experiments with negligibly close probability. (This follows from a standard hybrid argument.) Now, let $\tilde{M}^{V^*}$ be a machine who participates in the above experiments (playing the role of D). Machine $\tilde{M}^{V^*}$ works exactly like M^{V^*} except that instead of generating a prover-commitment to a string r_j by itself, it queries its oracle with r_j and uses the response c_j as if it is a commitment to r_j. (Note that we abuse notation here and ignore the fact that r_j can be from any iteration and any execution.) Now, on the one hand, if $\tilde{M}^{V^*}$ is involved in the first experiment (thereby receiving commitments to r_j), then it generates *exactly* the same distribution as M^{V^*}. On the other hand, if $\tilde{M}^{V^*}$ is involved in the second experiment (thereby receiving commitments to 0^n), then we claim that it generates a distribution that is statistically close to a real execution between P and V^*. The fact that the distribution is statistically close to a real execution requires justification because $\tilde{M}^{V^*}$ rewinds V^*. Intuitively, this holds because in both cases, the commitments received by V^* are to 0^n; furthermore, the rewinding carried out by $\tilde{M}^{V^*}$ is such that it does not skew the distribution of the transcript in any way. This is formally shown by induction over the fixed points. The base case refers to an empty transcript (this is as if a fixed point is placed before the simulation begins). Now, assume that indistinguishability holds until the ith fixed point is set; we now prove for the $i{+}1^{\text{th}}$ fixed point. In order to prove this, first note that the setting of a fixed point can be viewed as the following two-step process:

1. Sample the event that causes the next fixed point to be set in a real execution. There are at most 2 such events that can occur for every execution: 1 event for the case that the verifier aborts, and 1 event for the case that a potential iteration is the first to close.

2. Given the event that causes the next fixed point to be set, sample from all real transcripts that result in this event occurring.

The above "sampling" is according to the distribution defined by $V^*(x, z, r, t)$, where x, z, r are V^*'s common-input, auxiliary-input and random-tape, respectively, and t is the transcript up to the ith fixed point. Note that the distribution in question is dependent upon the prover's coins only (V^*'s coins r are fixed here). Now, if $\tilde{M}^{V^*}$ does not carry out any rewinding in setting the $i{+}1^{\text{th}}$ fixed point, then the portion of the transcript between the i and $i{+}1^{\text{th}}$ fixed point that is obtained, is distributed *exactly* like the transcript of a real execution between P and V^*. (Recall that in this case $\tilde{M}^{V^*}$ sends commitments to 0^n (just like the real prover) and furthermore, it uses a "real witness" for proving the proof of part 2.) We note that although the transcript is sampled directly, it implicitly follows the above described two-step process. Next, consider the case that rewinding

is carried out by $\tilde{M}^{V^*}$. In this case, there are two possibilities. First, $\tilde{M}^{V^*}$ may output failure; if this happens, then the resulting transcript is different from in a real execution. However, since this happens with at most negligible probability (by Claim 3.2), this causes at most a negligible difference. In contrast, if $\tilde{M}^{V^*}$ does not output failure, then the transcript that is output is either the same one as before rewinding (as when a failed fixed point is set), or the last one after rewinding (if $\tilde{M}^{V^*}$ succeeds in learning r_j). However, both of these transcripts are uniformly chosen from the set of transcripts in which the potential iteration in question is the first to close (i.e., the sampling is done exactly according to the distribution over all transcripts, conditioned on the event that causes the fixed point to be set). It follows that the transcript generated by $\tilde{M}^{V^*}$ in the second experiment is *statistically close* to a real transcript between P and V^* (the only difference is in the case that failure is output). In contrast, as shown above, the transcript generated by $\tilde{M}^{V^*}$ in the first experiment is exactly the same as by M^{V^*}. Therefore, by the indistinguishability of the two experiments, we conclude that the transcript distribution generated by M^{V^*} is computationally indistinguishable from the transcript generated in a real execution.

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 witness used for proving part 2 of the proofs. Specifically, M always uses the "real" witness for the statement being proved, whereas S always uses the witness that $c_j = \mathsf{Commit}(r_j)$. Thus, the indistinguishability of the distributions reduces to the zero-knowledge (or actually witness indistinguishability) property of the proof in part 2 of the protocol. (We also rely here on the fact that witness indistinguishability is preserved under concurrent composition [16].) This reduction is straightforward and is therefore omitted. ■

This completes the proof of Proposition 2. ■

Remark 3.1 – 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 3.3. Specifically, it holds that the number of iterations needs to be greater than or equal to $\kappa(n)$ times the number of *non-failed fixed points* that are placed by the simulator. In the context of bounded concurrent zero-knowledge, the number of non-failed fixed points equals 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 $m\kappa(n)$), and the number of non-failed fixed points required remains m only. This will become important when Protocol RKZK is used for obtaining bounded-concurrent secure two-party computation.

Remark 3.2 – reducing the length of the preamble. We also remark that the length of the preamble can be reduced to m, rather than $m\kappa(n)$ by employing an *expected polynomial-time* simulation strategy. The difference in the simulator is that when a potential iteration closes first, the verifier is rewound repeatedly until that same iteration closes first again. (This is in contrast to the above-described strategy where at most $2mn^2$ rewinding attempts are made.) We note that this must be done carefully in order to make sure that the simulator remains expected polynomial-time; techniques for such a simulation can be found in [20]. The reason that we use strict polynomial-time simulation (at the expense of additional rounds) is due to the fact that the zero-knowledge proof is used as a subprotocol in a larger protocol that utilizes "ideal calls" to zero-knowledge. Furthermore, this larger protocol has been proven secure for strict polynomial-time adversaries only. Now, if our zero-knowledge simulation ran in expected polynomial-time, then we would encounter the following problem. In order to prove the security of our entire protocol (i.e., the larger protocol combined with our zero-knowledge subprotocol), we first simulate the

zero-knowledge proofs, thereby obtaining an adversary for the larger protocol in a setting with ideal zero-knowledge calls. Next, we derive security by applying the proof of security for the larger protocol. Now, if the simulation of the zero-knowledge proof takes expected polynomial-time, then we would obtain an expected polynomial-time adversary for the larger protocol. However, as we have mentioned, this protocol has only been proven secure for strict polynomial-time adversaries. The proof of security would therefore not go through.

3.3 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 from the verifier during the generation protocol (this trapdoor will be used later on in the simulation). 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. In the definition below, we refer to $\mathsf{Ntime}(f(n))$ relations; a relation R is said to be in $\mathsf{Ntime}(f(n))$ if there exists a non-deterministic Turing machine that upon input (x, y), runs for at most $f(|x|)$ steps and outputs 1 if and only if $(x, y) \in R$. We now present the definition:

Definition 3 (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[15] *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) probabilistic 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) probabilistic 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.)*

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

(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 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). Then, of course, it holds that $M(c) = V^*(c)$ and so the simulator 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 $c = \mathsf{Commit}(0^n)$.*
  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. Loosely speaking, the relation consists of pairs (c, r) and (M, y) such that $c = \mathsf{Commit}(M)$ and $M(c, y) = r$. In other words, the relation is "hit" if the prover succeeds in committing to a machine that outputs the verifier's response to the commitment itself (as described in the motivation above). We note that the actual relation also includes h (the hash function from the protocol) and s (the random coins used in generating the commitment c to M). Formally, 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, $c = \mathsf{Commit}(h(M); s)$. 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 between the time that P sends the commitment c and V replies with $r \in_R \{0,1\}^{\ell(n)}$ 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.

4 A Special-Purpose Composition Theorem

In this section, we consider a model where the parties run a real protocol that uses ideal calls to a zero-knowledge proof of knowledge functionality (computed by a trusted third party). We then present a composition theorem that demonstrates that in the setting of m-bounded concurrency, a protocol in this model can be transformed into a different protocol for the real model (with no trusted party), such that the output distributions from the two protocols are essentially the same. This composition theorem is the central tool in our construction of m-bounded concurrent secure two-party computation.

4.1 The ZK-Hybrid Model

In general, a *hybrid model* is one where parties send protocol messages to each other (like in the real model), but also have access to a trusted party that computes some functionality (like in the ideal model). Thus, this model is a hybrid of the real and ideal models. A protocol that is designed for a hybrid model contains two types of messages: real messages that are sent directly between the parties, and ideal messages that are sent between the parties and the trusted third party.

In the ZK-hybrid model, the parties have access to a trusted party who computes the following ideal zero-knowledge functionality:

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

where R is an NP-relation. 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 because the prover must provide an actual witness to the trusted party. 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 $(v, \perp) \notin R$. Thus, upon receiving $(v, \perp)$, the trusted party always sends $(v, 0)$ to the verifier.

Let $p(n)$ be a polynomial and let $\overline{x}, \overline{y} \in (\{0,1\}^n)^{p(n)}$ 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 protocol Π also contains ideal messages which are dealt with by the trusted party who computes the above zero-knowledge functionality.

4.2 Motivation for the Composition Theorem

Informally speaking, the composition theorem states that in the setting of m-bounded concurrent composition, 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. In other words, the ideal zero-knowledge calls can be replaced by a real protocol, and the result is the same. Jumping ahead, this basically means that it suffices to obtain a secure protocol for the ZK-hybrid model, and we can infer the existence of a secure protocol for the real model. As we will see in Section 5, such protocols already exist, and so our goal is fulfilled.

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 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 in the hybrid model, the zero-knowledge proofs are simulated, and such simulation essentially constitutes "cheating". Therefore, if the adversary could cheat by mauling the simulated proofs, this would mean that the result of a real execution (where the adversary cannot cheat) and a hybrid execution (where it can) would be different. 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; that is, it runs in the context of *self composition* only. In contrast, here the zero-knowledge protocol is run concurrent to itself *and* to other protocols, as in the case of *general composition*. Therefore, one cannot automatically claim that the zero-knowledge property holds here. In fact, rewinding techniques (used in all black-box zero-knowledge protocols) are very problematic in this context. In order to see this, notice that 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 protocol (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 prove statements with the non-black-box zero-knowledge protocol of [1] (see Section 3.3), while the other party proves statements with the black-box zero-knowledge protocol of [37] (see Section 3.2). Loosely speaking, it turns out that these protocols are both non-malleable (or more accurately, simulation-sound) 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 of concurrency with other protocols is solved as follows. First, the simulation strategy for the protocol of [1] does not use rewinding. 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 simulation strategy of the protocol of [37] does use rewinding, which as we have described is problematic here. We solve this problem by using the simulation strategy demonstrated in Section 3.2. 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 non-failed fixed points that need to be 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. Then, since rewinding only takes place between fixed points, this ensures

that no rewinding behind a Π'-message is necessary. This strategy is based on the methodology used by [21] to solve the problem of simulating zero-knowledge while external messages may be sent.

4.3 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.

Theorem 4 *Let Π' be a two-party protocol for the* ZK-hybrid model *and let $m(\cdot)$ be a polynomial. 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 polynomial-time adversary $\mathcal{A}'$ running Π' in the* ZK-hybrid *model such that for every polynomial $p(n)$,*

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

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' (recall that in Π' this involves interaction with the trusted party only). 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 3.1. We denote this protocol by POK. Recall that in this system of arguments of knowledge, both the prover and verifier play the role of prover in zero-knowledge proofs of *membership*; we call these the $\mathsf{subproofs}$. Now, all the zero-knowledge subproofs in which P_1 is the prover are proven using the black-box concurrent zero-knowledge protocol of [37], described in Section 3.2; denote this protocol by RKZK. Furthermore, all the zero-knowledge subproofs in which P_2 is the prover are proven using the concurrent zero-knowledge protocol of [1], described in Section 3.3; denote this protocol by BZK. That is, when P_1 plays the prover in the zero-knowledge proof of knowledge POK, then P_2 uses Protocol BZK when proving its subproof from the commit-with-extract scheme and P_1 uses Protocol RKZK when proving its subproof in part 2 of POK. Conversely, when P_2 plays the prover in the zero-knowledge proof of knowledge POK, then P_1 uses Protocol RKZK when proving its subproof from the commit-with-extract scheme and P_2 uses Protocol BZK when proving its subproof in part 2 of POK. *The important thing to remember is that all the subproofs in which P_1 plays the prover use Protocol* RKZK*, and all the subproofs in which P_2 plays the prover use Protocol* BZK. We note that the proof of knowledge POK and the subproofs RKZK 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 protocol POK:

- *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 POK 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 POK 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 RKZK and BZK protocols are used as subproofs in POK. Both of these protocols are parameterized: the RKZK 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 RKZK the parameter k is set to $m\kappa(n)$, 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 the 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 total length of the messages in the proof of knowledge *without* the subproofs RKZK and BZK. The parameters for the zero-knowledge subproofs are set as follows. Let $\kappa(\cdot)$ be any super-constant (and sub-linear) function. Then, in RKZK we set

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

and in BZK we set

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

In other words, the number of iterations in the RKZK preamble is $\zeta m\kappa(n)$ plus $m\kappa(n)$ times the combined number of rounds in Π excluding the RKZK subproofs; this combined number of rounds comes to $\mathsf{rounds}(\Pi') + \zeta\cdot\mathsf{rounds}(\mathsf{POK}) + \zeta\cdot\mathsf{rounds}(\mathsf{BZK})$. (The first $\zeta m\kappa(n)$ iterations are used to cover the ζm executions of RKZK, as discussed in Remark 3.1. The number of additional iterations is fixed so that there are $\kappa(n)$ iterations for every non-RKZK message that is sent in all m executions of the protocol.)

Likewise, the length $\ell(n)$ of the message r in BZK is set to $2\zeta mn^2$ (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 m executions of Π. We remark that there is no problem setting the parameters in such a way. That is, although $\ell(n)$ is dependent on $\mathsf{length}(\mathsf{RKZK})$, the number of rounds in BZK is *not* dependent on RKZK. 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 $O(m\kappa(n))$, where $\kappa(n)$ is any super-constant function (e.g., $\kappa(n) = \log\log n$).

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.

4.3.1 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 internal and 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, possibly obtaining a valid witness w for v. (Actually, it runs the *witness-extended emulator* for POK, and obtains both a transcript of an execution along with a value that is possibly a witness. See the short discussion in Section 3.1 on witness-extended emulation.) 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 RKZK 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 3.1.) As we have mentioned, from this extraction procedure (or, actually, witness-extended emulation), $\mathcal{A}'$ receives some string w, which is possibly a witness for v, and a transcript of an execution of the proof. $\mathcal{A}'$ verifies that the transcript constitutes an accepting proof (this can be publicly verified by the construction of Protocol POK). 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 RKZK 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 3.1.)
  (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 S^* of [1], as described in Section 3.3. Notice that in the above simulation, all of the subproofs in which $\mathcal{A}$ is the prover are RKZK, and all of the subproofs in which $\mathcal{A}$ is the verifier are BZK. 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 RKZK 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 RKZK and BZK executions. We do this by defining an ideal zero-knowledge functionality that is a proof of membership, and not a proof of knowledge. We then show that replacing the RKZK and BZK subproofs by this ideal functionality makes at most a negligible difference. This enables us to deal with the RKZK and BZK subproofs separately.

The hybrid experiment. The hybrid experiment involves an ideal zero-knowledge *proof of membership* functionality, parameterized by an NP-language L, and defined by the following:

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

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 the statement 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 memberzk-hybrid.

Step 1 – replacing BZK with an ideal zero-knowledge proof of membership. We now define a protocol $\hat{\Pi}$ that works in the memberZK-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 zero-knowledge proof of membership functionality. Subproofs given by P_1 using RKZK are unmodified.

We now show that for every real-model adversary $\mathcal{A}$ for Π, there exists an adversary $\hat{\mathcal{A}}$ for $\hat{\Pi}$, such that $\hat{\mathcal{A}}$ interacts with P_2 in the memberZK-hybrid model and 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 zero-knowledge proof of membership functionality. Thus, the proof of indistinguishability is based on the zero-knowledge (or simulatability) 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 RKZK 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 zero-knowledge proof of membership functionality, it simulates an execution of BZK with $\mathcal{A}$ as the verifier. We now prove that

$$\left\{\textsc{memberzk-hybrid}^m_{\hat{\Pi},\hat{\mathcal{A}}}(\overline{x}, \overline{y}, z)\right\} \stackrel{c}{\equiv} \left\{\textsc{real}^m_{\Pi,\mathcal{A}}(\overline{x}, \overline{y}, z)\right\} \tag{1}$$

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 3.3, the simulator for BZK works on the condition that the total length of all the messages sent by the prover (i.e., P_2) from the time that it sends the commitment c in BZK until the time that P_1 returns $r \in \{0,1\}^{\ell(n)}$ is less than $\ell(n) - n$. In order to see that this holds, recall that in the setting of m-bounded concurrency, the scheduling is such that from the time that any given execution begins until it terminates, messages from at most m different executions are sent. Therefore, the length of all messages sent by P_2 from the time that it sends c until P_1 replies with r is upper-bound by m times the length of all the messages sent by P_2 in an execution of Π. However, this follows immediately from the way $\ell(n)$ was chosen.

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

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

$$\left\{\textsc{memberzk-hybrid}^m_{\tilde{\Pi},\tilde{\mathcal{A}}}(\overline{x},\overline{y},z)\right\} \stackrel{\mathrm{c}}{\equiv} \left\{\textsc{memberzk-hybrid}^m_{\hat{\Pi},\hat{\mathcal{A}}}(\overline{x},\overline{y},z)\right\} \tag{2}$$

First, assume that $\tilde{\mathcal{A}}$ accepts a (false) RKZK subproof for $v \notin L$ from $\hat{\mathcal{A}}$ with at most negligible probability. In this case, we show that the output distributions of an execution of $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$ and an execution of $\hat{\Pi}$ with $\hat{\mathcal{A}}$ are statistically close. This can be seen as follows. In verifying the RKZK subproofs, both $\tilde{\mathcal{A}}$ and P_2 behave identically; specifically, they both follow the honest verifier strategy. However, if $\tilde{\mathcal{A}}$ accepts a subproof for a statement v from $\hat{\mathcal{A}}$ in $\tilde{\Pi}$, then it sends v to the trusted party computing the zero-knowledge proof of membership functionality. The trusted party then sends $(v,1)$ to P_2 if $v \in L$, and $(v,0)$ otherwise. Finally, P_2 writes whatever it receives from the trusted party directly on its transcript. This describes the process in $\tilde{\Pi}$. In contrast, in $\hat{\Pi}$, if P_2 accepts such a subproof from $\hat{\mathcal{A}}$, then it immediately writes $(v,1)$ on its transcript in $\hat{\Pi}$, without any other conditions. Therefore, a difference in the output distributions can occur if $\tilde{\mathcal{A}}$ accepts a subproof for v, but $v \notin L$. In such a case, P_2 would write $(v,1)$ on its transcript in $\hat{\Pi}$, but would write $(v,0)$ on its transcript in $\tilde{\Pi}$. However, by the above assumption, this event occurs with at most negligible probability. Therefore, the output distributions are negligibly close. (Another possible difference can occur if $\tilde{\mathcal{A}}$ rejects a subproof for v, but $v \in L$. However, if $\tilde{\mathcal{A}}$ rejects a subproof for v, then it sends $v \circ \perp$ to the trusted party, who then always forwards $(v,0)$ to P_2, irrespective of whether or not $v \in L$. Therefore, in this case, party P_2 writes $(v,0)$ on its transcript in both $\tilde{\Pi}$ and $\hat{\Pi}$.)

It remains to show that $\tilde{\mathcal{A}}$ accepts a proof for $v \notin L$ from $\hat{\mathcal{A}}$ with at most negligible probability. That is, Eq. (2) follows from the *soundness* of RKZK. Since the soundness of RKZK is uncondi-

tional, 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 RKZK 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 zero-knowledge proof of membership calls, because P^* is not computationally bounded.) Now, since $\tilde{\mathcal{A}}$ verifies RKZK 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 RKZK 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 memberZK-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:[16]

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

Before proving Eq. (3), we describe $\mathcal{A}'$ in terms of $\tilde{\mathcal{A}}$. That is, our initial description of $\mathcal{A}'$ is such that it internally incorporates the real-model adversary $\mathcal{A}$ and runs all the simulations. However, we have considered these same simulations separately by $\hat{\mathcal{A}}$ and $\tilde{\mathcal{A}}$. Therefore, we can rewrite $\mathcal{A}'$ as follows: Adversary $\mathcal{A}'$ internally incorporates $\tilde{\mathcal{A}}$ and simulates the POK messages that $\tilde{\mathcal{A}}$ expects to receive in $\tilde{\Pi}$. $\mathcal{A}'$ also plays the trusted party for the zero-knowledge proof of membership functionality that replaces the subproofs in POK (we will describe below how this emulation of the trusted party is carried out). All other Π' messages are forwarded by $\mathcal{A}'$ to the external party P_2. Now, when $\mathcal{A}'$ receives $(v, 1)$ from its trusted party (recall that $\mathcal{A}'$ works in the ZK-hybrid model), then it runs the simulation strategy of POK. Likewise, when $\tilde{\mathcal{A}}$ proves a statement v in an execution of POK in $\tilde{\Pi}$, then $\mathcal{A}'$ runs the extraction strategy (or actually the witness-extended emulation), and obtains a witness w. If the execution of POK was accepting, then $\mathcal{A}'$ sends (v, w) to its trusted party; otherwise, it sends $(v, \perp)$.

In the above simulation and extraction, it is necessary for $\mathcal{A}'$ to simulate the trusted party for the zero-knowledge proof of membership functionality. However, this trusted party may need to run in time that is super-polynomial in order to verify if some statement v' is in the language L or not. Therefore, $\mathcal{A}'$ simply works as follows: if $\tilde{\mathcal{A}}$ sends v' to its trusted party, then $\mathcal{A}'$ assumes

[16] 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 ideal zero-knowledge proof of membership 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 $v \in L$.

that $v' \in L$ and continues the simulation/extraction strategy under this assumption. On the other hand, if $\tilde{\mathcal{A}}$ sends $v' \circ \perp$ to its trusted party, then $\mathcal{A}'$ continues as if $v' \notin L$. (Note that $\tilde{\mathcal{A}}$ may send $v' \circ \perp$ even if $v' \in L$; however, the simulation and extraction strategies are the same for this case and when $v' \notin L$.) This completes the description of $\mathcal{A}'$.

Now, by Lemma 3.1, if the subproofs of Protocol POK have negligible soundness error and can be simulated (i.e., are zero-knowledge), then the simulation and extraction strategies for POK succeed. The zero-knowledge property of the subproofs is straightforward because here $\mathcal{A}'$ emulates the trusted party who just sends $(v', 1)$ every time. (When simulating the subproofs, the case in question is where the honest P_2 is the prover; therefore, only correct statements v' are sent to the trusted party and so $\tilde{\mathcal{A}}$ always receives (v', b) with $b = 1$.) On the other hand, the soundness property of the subproofs is due to the fact that by the construction of $\tilde{\mathcal{A}}$, we know that the probability that it sends v' (and not $v' \circ \perp$) to the trusted party for $v' \notin L$ is negligible. We can therefore apply Lemma 3.1 and we have that the simulation and extraction strategies for POK succeed. In other words, the outcome of an execution of Π' with $\mathcal{A}'$ is computationally indistinguishable from the outcome of an execution of $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$, proving Eq. (3). This proof is somewhat lacking due to the fact that Lemma 3.1 is informally stated, and thus so is its application. In Appendix A, we state and prove a formal version of Lemma 3.1, and show how it can be used to obtain Eq. (3).

The theorem for the case that P_1 is corrupt follows by combining Equations (1), (2) and (3).

Before proceeding to the case that P_2 is corrupted, we present a summary of the flow of the above proof for the case that P_1 is corrupted; see Table 1.

Step	Protocol	Zero-knowledge POK and subproofs
	Π	All *real*: POK, BZK and RKZK
1	$\hat{\Pi}$	BZK replaced with ideal memberZK; POK and RKZK real
2	$\tilde{\Pi}$	BZK, RKZK replaced with ideal memberZK, POK real
3	Π'	All *ideal*: POK, BZK and RKZK replaced with ideal-ZK

Table 1: Proof outline for the case that P_1 is corrupted

4.3.2 Party P_2 is Corrupted

We construct the ZK-hybrid model adversary $\mathcal{A}'$ in a similar way to the previous case. Before describing $\mathcal{A}'$, recall that P_1 proves all of its subproofs with RKZK, and P_2 proves all of its subproofs with BZK. Adversary $\mathcal{A}'$ internally incorporates $\mathcal{A}$ and externally forwards all Π'-messages between $\mathcal{A}$ and P_1. In addition:

- 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, possibly obtaining a valid witness w for v. (Actually, it runs the witness-extended emulator for POK, and obtains both a transcript of an execution and a possible witness; see Section 3.1.) This involves simulating Protocol RKZK 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 3.1.) As we have mentioned, from this extraction procedure, or actually witness-extended emulation, $\mathcal{A}'$ receives some string w which

is possibly a witness for v, and a transcript of an execution of the proof. $\mathcal{A}'$ verifies that the transcript constitutes an accepting proof. 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 RKZK 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 3.1.

We now describe how $\mathcal{A}'$ carries out the simulation of RKZK. The simulation uses a very similar strategy to that described in the proof of Proposition 2 (see Section 3.2). In our description here, we will use notation and terminology taken from the proof of Proposition 2 (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-RKZK 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 RKZK subproofs.)

2. An execution of Protocol RKZK is aborted.

3. 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 2. Specifically, whenever a fixed point of type (1) or (2) above is sent, $\mathcal{A}'$ fixes the transcript until (and including) the current message. Furthermore, if the fixed point is due to an execution of RKZK being aborted, then this execution is satisfied. Therefore, $\mathcal{A}'$ removes the execution from the set of unsatisfied executions. The treatment of a fixed point of type (3) is different. That is, whenever a potential iteration closes, $\mathcal{A}'$ carries out $2mn^2$ rewinding attempts: in each one, $\mathcal{A}'$ rewinds $\mathcal{A}$ back to the point that the iteration opened and provides it with a commitment to r_j instead of 0^n. $\mathcal{A}'$ then continues, hoping that this iteration will once again close before any non-RKZK messages are sent or any executions of RKZK are aborted. If none of the $2mn^2$ rewinding attempts succeed, then $\mathcal{A}'$ sets a fixed point, records it as a failed fixed point, and continues. If at any stage of the simulation $\kappa(n)$ consecutive failed fixed points are set, then $\mathcal{A}'$ outputs failure and halts. Likewise, the other instructions that cause the RKZK-simulator S to output failure or unsatisfied are also followed by $\mathcal{A}'$; see Section 3.2 for more details. This completes the description of $\mathcal{A}'$.

We begin by noting that $\mathcal{A}'$ runs in time that is polynomial in n. Specifically, the overall number of rewinding attempts is bounded by $2mn^2$ times the overall number of fixed points, and all other work also takes polynomial time. Next, we prove that the output distribution of an execution of Π' with $\mathcal{A}'$ is computationally indistinguishable from the output distribution of an execution of Π with $\mathcal{A}$. We prove this using the same hybrid experiment as in the case that P_1 is corrupted.

Step 1 – replacing RKZK with an ideal zero-knowledge proof of membership. We define the protocol $\tilde{\Pi}$ for the memberZK-hybrid model in a similar way as for the case that P_1 is corrupted. That is, in this protocol, ideal calls to the zero-knowledge proof of membership functionality are used instead of running the RKZK subproofs, but the rest (i.e., POK and BZK) stays the same. We now show that for every real-model adversary $\mathcal{A}$ for Π, there exists an adversary

$\tilde{\mathcal{A}}$ for $\tilde{\Pi}$ in the memberZK-hybrid model, 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-RKZK 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 zero-knowledge proof of membership functionality, it simulates a RKZK execution for $\mathcal{A}$ using exactly the same strategy described above for $\mathcal{A}'$. We now prove that,

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

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 2. We first claim that $\tilde{\mathcal{A}}$ outputs failure with only negligible probability:

Claim 4.1 *The probability that $\tilde{\mathcal{A}}$ outputs* failure *is negligible.*

The proof of this claim is almost identical to the proof of Claim 3.2. (Recall that there are two events that can cause $\tilde{\mathcal{A}}$ to output failure. In the first, for some commitment c_j, $\mathcal{A}$ generates two different decommitments $r_j \neq r'_j$ during rewinding. In the second, $\kappa(n)$ consecutive failed fixed points are set by $\tilde{\mathcal{A}}$. Regarding the first event, the only difference here from the proof of Claim 3.2 is that there are ζm executions of RKZK 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 input vectors $\overline{x}$ and $\overline{y}$. Regarding the second event, the proof remains the same. That is, we first consider the case that $p^j_k \geq 1/mn$; in this case the probability that none of the $2mn^2$ rewinding attempts succeeds is negligible. Next, we consider the case that $p^j_k < 1/mn$ and use the same proof to show that the probability that such an iteration was chosen is at most $1/n$. It therefore follows that $\kappa(n)$ consecutive iterations are encountered with at most negligible probability.)

Next, we claim that $\tilde{\mathcal{A}}$ never outputs unsatisfied:

Claim 4.2 *The probability that $\tilde{\mathcal{A}}$ outputs* unsatisfied *equals* 0.

Proof: As in the proof of Claim 3.3, it suffices to show that if there are not $\kappa(n)$ consecutive failed fixed points, then every execution of RKZK must have been satisfied. Here, this follows from the fact that the number of iterations in the preamble of RKZK is $\zeta m\kappa(n)$ plus the number of rounds in the rest of the protocol times $\kappa(n)$. Notice that $\tilde{\mathcal{A}}$ sets at most one non-failed fixed point for every execution of RKZK (ζm in total) and one non-failed fixed point for every message from the rest of the protocol (consisting of Π', POK and BZK). That is, the maximum number of non-failed fixed points set by $\tilde{\mathcal{A}}$ equals $\zeta m + m \cdot (\mathsf{rounds}(\Pi') + \mathsf{rounds}(\mathsf{POK}) + \mathsf{rounds}(\mathsf{BZK}))$, and if this number of non-failed fixed points are set, then all executions of RKZK are guaranteed to have been satisfied.

Now, recall that at most one iteration of an unsatisfied execution can begin between every two fixed points. Therefore, if part 2 of an unsatisfied execution is reached, it must be that $\zeta m\kappa(n) + m\kappa(n) \cdot (\mathsf{rounds}(\Pi') + \mathsf{rounds}(\mathsf{POK}) + \mathsf{rounds}(\mathsf{BZK}))$ fixed points were set. Assuming that there are not $\kappa(n)$ consecutive failed fixed points, this means that at least $\zeta m + m \cdot (\mathsf{rounds}(\Pi') + \mathsf{rounds}(\mathsf{POK}) + \mathsf{rounds}(\mathsf{BZK}))$ non-failed fixed points were set. However, this equals the *total* number of non-failed fixed points that can be set, in which case, all executions are RKZK are guaranteed to have been satisfied. We conclude that when part 2 of an execution of RKZK is reached, it must already have been satisfied. ■

Having established the above, the indistinguishability of the distributions follows with an analogous hybrid argument to the proof of Claim 3.4. Specifically, we define a hybrid adversary/simulator M who receives the witnesses for each proof being proved (from the honest P_1) and runs part 2 of the proof of RKZK 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 RKZK). The proof continues as for Claim 3.4. This completes the proof of Eq. (4).

Step 2 – replacing BZK with an ideal zero-knowledge proof of membership. Again, as in the case that P_1 is corrupted, we define a protocol $\hat{\Pi}$ in which both the subproofs RKZK and BZK are proven using the ideal zero-knowledge proof of membership functionality (but the POK messages remains unchanged). We show that for every adversary $\tilde{\mathcal{A}}$ for $\tilde{\Pi}$ there exists an adversary $\hat{\mathcal{A}}$ for $\hat{\Pi}$ such that the output distribution of the two executions are indistinguishable. Adversary $\hat{\mathcal{A}}$ internally incorporates $\tilde{\mathcal{A}}$ and externally forwards all Π'-messages and POK-messages between $\tilde{\mathcal{A}}$ and P_1. In addition, $\hat{\mathcal{A}}$ forwards all ideal messages from the trusted party who computes the zero-knowledge proof of membership functionality to $\tilde{\mathcal{A}}$ (recall that $\tilde{\mathcal{A}}$ works in a model where it only *receives* ideal messages from the trusted party). 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 zero-knowledge proof of membership functionality; otherwise, it sends $v \circ \perp$. We now prove that

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

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 3.3, 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 RKZK. The key point here is that $\hat{\mathcal{A}}$ never rewinds $\tilde{\mathcal{A}}$. 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 memberZK-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).[17] Now, when $\tilde{\mathcal{A}}$ reaches the selected BZK 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 BZK 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 BZK supplied by $\tilde{\mathcal{A}}$, we conclude that the probability that P^* successfully

[17] The problem that arises here is that the zero-knowledge proof of membership functionality cannot be efficiently computed for hard languages L. Note that any proofs provided by P_1 are fine, because P_1 is honest and so only sends proofs of correct statements $v \in L$. However, when $\tilde{\mathcal{A}}$ sends a value v to the trusted party, $\hat{\mathcal{A}}$ cannot know whether or not $v \in L$.

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 BZK, 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 zero-knowledge proof of membership 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 zero-knowledge proof of membership 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 4. ■

Remark: We call attention to the fact that in the proof of Theorem 4, the ZK-hybrid model adversary $\mathcal{A}'$ either verifies RKZK and simulates BZK (when P_1 is corrupt), or simulates RKZK 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.

4.4 Generalizing the Composition Theorem

Theorem 4 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 4). Thus, we really demonstrate the feasibility of obtaining m-bounded concurrent composition in the real model of *arbitrary* protocols that are all designed in the ZK-hybrid model.

5 Obtaining Bounded Concurrent Two-Party Computation

In this section, we show that by Theorem 4, 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 proof of knowledge functionality $((v,w),\lambda) \mapsto (\lambda,(v,R(v,w))$. Security in the ZK-hybrid model is defined in the natural way. That is,

Definition 5 (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 probabilistic polynomial-time* ZK-hybrid *adversary $\mathcal{A}'$ running Π' there exists a non-uniform probabilistic polynomial-time ideal-model adversary $\mathcal{S}$ such that for every polynomial $p(n)$,*

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

Security in the ZK-hybrid model suffices. An important corollary of Theorem 4 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.

Corollary 6 (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. 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 protocol Π in the corollary statement is the one that is guaranteed to exist by the transformation of Π' that is stated in Theorem 4. The proof that the real-model protocol Π securely computes f works by showing the existence of an ideal-model adversary $\mathcal{S}$ for every real-model adversary $\mathcal{A}$. Let $\mathcal{A}$ be a probabilistic polynomial-time real-model adversary for Π. Then, by Theorem 4, there exists a probabilistic 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 probabilistic polynomial-time adversary $\mathcal{A}'$ there exists a probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model such that

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

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

Bounded concurrent secure computation in the plain model. By Corollary 6, 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 under m-bounded concurrent composition. However, such a protocol is already known to exist. In particular, assuming the existence of enhanced trapdoor permutations, it has been shown that any two-party functionality can be securely computed in the

ZK-hybrid model under the definition of universal composability [13].[18] Since universal composability implies m-bounded concurrent composition, we obtain the following theorem:

Theorem 7 (Theorem 1 – restated): *Assume that enhanced trapdoor permutations and collision resistant hash functions exist. Then, for any probabilistic polynomial-time 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 $m\kappa(n)$ times the number of rounds in the protocol of [13]. We remark that the construction of [13] for the case of static adversaries can be made constant-round by using the semi-honest protocol of Yao [39]. Therefore, the round complexity of our protocol is $O(m\kappa(n))$.

Acknowledgements

We would like to thank Ran Canetti, Jonathan Katz, Rafael Pass, Tal Rabin, and Alon Rosen for helpful discussions and comments.

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. *SIAM Journal on Computing,* 33(4):783–818, 2004.

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

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

[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.

[18] Technically, the results of [13] do not hold for *any* efficient functionality, but for a subset that they call "well-formed". However, in the case of static adversaries, the only issue is that the functionality should behave independently of the set of corrupted parties (in the setting of universal composability, this information on who is corrupted is provided to the functionality). In our model, the trusted party is anyway not given this information and therefore there is no limitation on the functionalities that can be computed.

[9] R. Canetti. On Universally Composable Notions of Security for Signature, Certification and Authentication. Available at the Cryptology ePrint Archive `http://eprint.iacr.org/`, Report #2003/239, 2003.

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

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd STOC*, pages 416–426, 1990.

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

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

[19] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, 2004.

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

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

[22] 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].

[23] 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.

[24] J. Kilian and E. Petrank. Concurrent and Resettable Zero-Knowledge in Poly-logarithmic Rounds. In 33*rd STOC,* pages 560–569, 2001.

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

[26] Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. *Journal of Cryptology*, 16(3):143–184, 2003.

[27] Y. Lindell. Bounded-Concurrent Secure Two-Party Computation Without Setup Assumptions. In 35*th STOC*, pages 683–692, 2003.

[28] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. In 44*th FOCS*, pages 394–403, 2003.

[29] Y. Lindell. *Composition of Secure Multi-Party Protocols – A Comprehensive Study.* Lecture Notes in Computer Science Vol. 2815, Springer-Verlag, 2003.

[30] Y. Lindell. Lower Bounds for Concurrent Self Composition. In the *1st Theory of Cryptography Conference* (TCC), Springer-Verlag (LNCS 2951), pages 203–222, 2004.

[31] Y. Lindell, A. Lysysanskaya and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In 34*th STOC*, pages 514–523, 2002.

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

[33] R. Pass. Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority. To appear in the 36*th STOC,* 2004.

[34] R. Pass and A. Rosen Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. In 44*th FOCS*, pages 404–413, 2003.

[35] B. Pfitzmann and M. Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In *7th ACM Conference on Computer and Communication Security*, pages 245–254, 2000.

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

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

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

A A Formal Proof of Equation (3) in Section 4.3

The proof of Theorem 4 relies on Eq. (3), both for the case that P_1 is corrupted and for the case that P_2 is corrupted. However, our proof that Eq. (3) holds relies on Lemma 3.1 which is informally stated. A formal statement of this lemma and its proof can be derived from the proofs of [4]. However, the lemma is actually not proved there. In this appendix, we provide a formal proof of Eq. (3). This mainly involves providing a proof that Protocol 1 from [4] is a system of zero-knowledge arguments of knowledge in the memberZK-hybrid model. Actually, we prove something stronger. That is, loosely speaking, we prove that Protocol 1 securely realizes the ideal zero-knowledge proof of knowledge functionality in the memberZK-hybrid model, even when it is used as a subprotocol in some larger protocol Π that runs in the setting of m-bounded concurrency.

In our presentation here, we recap on some of the basic concepts used. However, we assume that the reader is familiar with Sections 3.1 and 4 (and so much of the motivation and high-level ideas are not repeated). We begin by recalling the memberZK and ZK-hybrid models, as defined in Section 4.

The memberZK-hybrid model. Recall that in a hybrid model, the parties interact with each other as in the real model. However, in addition to this interaction, they have access to a trusted party who computes some functionality for them. In the memberZK-hybrid model, this functionality is defined as follows. Let L be an NP-language. Then, the trusted party receives a statement v from the proving party and sends $(v, \chi_L(v))$ to the verifying party, where $\chi_L(v) = 1$ if and only if $v \in L$. That is, the functionality computed by the trusted party is:

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

Note that if $L \notin \mathcal{BPP}$, this functionality is not efficiently computable. Nevertheless, it suffices for our needs here. We also recall that if the statement received by the trusted party is of the form $v \circ \perp$, then it always sends $(v, 0)$ to the verifying party, irrespective of whether or not $v \in L$. This is needed in order to model the situation that a (cheating) prover may cause the verifier to reject a proof, even if the statement is true.

The ZK-hybrid model. The ZK-hybrid model is similar to the memberZK-hybrid model, except that the functionality is a *proof of knowledge*, and not just a proof of membership. This also means that the functionality can be efficiently computed. Let R be an NP-relation. Then, the ideal zero-knowledge proof of knowledge functionality is defined as follows:

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

Recall that we define a special symbol $\perp$ such that for every relation R, it holds that $R(v, \perp) = 0$. Again, this is used to model the case that a verifier rejects a true statement.

The protocol. We now rewrite Protocol 1 so that it is cast in the memberZK-hybrid model. We also combine the commit-with-extract part directly into the protocol, with a slight modification. Specifically, we have the prover choose the trapdoor permutation after it receives the commitment from the verifier (unlike the protocol description in Section 3.1 where the trapdoor permutation is chosen before the "coin-tossing" phase). This simplifies the proof to some extent, but is not necessary.

Protocol 4 (zero-knowledge argument of knowledge for $R \in \mathcal{NP}$ in the memberZK-hybrid model):

- **Common Input:** v
- **Auxiliary input to prover:** *w such that $(v, w) \in R$. For simplicity, assume that $|v| = |w| = n$.*
- **Part 1 – Commit-with-Extract:** *The prover commits to its auxiliary-input witness w.*
  1. The parties run a coin-tossing protocol:
     (a) *The verifier chooses a random string $r_1 \in_R \{0,1\}^n$ and sends $c = \mathsf{Commit}(r_1)$ to the prover (using any perfectly-binding commitment scheme* Commit*).*
     (b) The prover chooses an enhanced trapdoor permutation: *The prover chooses an enhanced trapdoor permutation f along with its trapdoor t and sends f to the verifier.*[19] *The verifier checks that f is a permutation, and aborts if not.*
     (c) *The prover chooses a random string $r_2 \in_R \{0,1\}^n$ and sends r_2 to the verifier.*
     (d) *The verifier sends r_1 to the prover (without decommitting).*
     (e) *The verifier proves that r_1 is the value that it indeed committed to. That is, the verifier sends (c, r_1) to the trusted party computing the zero-knowledge proof of membership functionality for the language $\{(c, r) \mid c = \mathsf{Commit}(r)\}$.*
     (f) *The prover receives $((c, r_1), \sigma)$ from the trusted party and aborts unless $\sigma = 1$.*
     (g) *The output of the coin-tossing phase is $r = r_1 \oplus r_2$.*
  2. The actual commitment: *Let b be a hard-core bit of f. Then, the prover computes the sequence $\tilde{r} = b(f^{-n}(r))b(f^{-n+1}(r)) \cdots b(f^{-1}(r))$.*
     The prover sends the value $\tilde{c} = \tilde{r} \oplus w$ to the verifier.
- **Part 2 – Proof of Correctness:** *The prover proves to the verifier that it committed to a valid witness w for v in the previous step:*
  1. *Define the language*
     $$L = \left\{ (v, \tilde{c}, f, r) \;\middle|\; R(v, \tilde{c} \oplus \tilde{r}) = 1 \text{ where } \tilde{r} = b(f^{-n}(r))b(f^{-n+1}(r)) \cdots b(f^{-1}(r)) \right\}$$
     Then, the prover sends $(v, \tilde{c}, f, r)$ to the trusted party computing the zero-knowledge proof of membership functionality.
  2. *The verifier accepts if and only if it receives $((v, \tilde{c}, f, r), 1)$ from the trusted party, where v is its common input, and $(\tilde{c}, f, r)$ are as generated in part 1.*

Non-abusing adversaries. For the sake of clarity, we use the notion of non-abusing adversaries, introduced by [34]. A non-abusing adversary works in the memberZK-hybrid model and is such that it sends a false statement $v \notin L$ to the trusted party that is not of the form $v' \circ \perp$ with at most negligible probability. That is, such an adversary attempts to "lie" with at most negligible probability. Note that when non-abusing adversaries are considered, the functionality can be efficiently computed. Specifically, there is no need to check whether or not $v \in L$; all that is necessary is to check whether or not the statement is of the form $v \circ \perp$. (We assume that honest provers also only send correct statements.)

[19] For the sake of simplicity, we assume that the protocol uses *certified* [15] enhanced trapdoor permutations (for which the verifier can efficiently verify that f is indeed a permutation). However, actually any enhanced trapdoor permutation can be used by having the prover prove that f is a permutation; see [4]. (Of course, here this proof is carried out using the zero-knowledge proof of membership functionality.)

We are now ready to present the formal analog of Lemma 3.1 that will then be used to derive Eq. (3).

Lemma A.1 *Let Π' be a two-party protocol that is designed in the* ZK-hybrid model *and let $m(\cdot)$ be a polynomial. Let Π be a two-party protocol that is obtained from Π' by replacing the ideal zero-knowledge calls with Protocol 4 that is designed in the* memberZK-hybrid model. *Then, assuming the existence of enhanced trapdoor permutations, for every probabilistic polynomial-time* non-abusing adversary $\mathcal{A}$ *for Π there exists a probabilistic polynomial-time adversary $\mathcal{A}'$ for Π' such that*

$$\left\{\text{ZK-HYBRID}^m_{\Pi',\mathcal{A}'}(\overline{x},\overline{y},z)\right\} \stackrel{c}{\equiv} \left\{\text{MEMBERZK-HYBRID}^m_{\Pi,\mathcal{A}}(\overline{x},\overline{y},z)\right\} \tag{9}$$

where the ensembles in Eq. (9) *are over $n \in \mathsf{N}$, $\overline{x},\overline{y} \in (\{0,1\}^n)^{p(n)}$ and $z \in \{0,1\}^*$.*

Proof: Much of the technical content in the proof of this lemma is the same as in [4]. The main differences are due to the model and definitions used. First, Lemma A.1 is stated in a setting of composition, whereas [4] relates to the stand-alone setting only. Second, here we construct Protocol 4 in the memberZK-hybrid model and show that it securely realizes the ideal zero-knowledge proof of knowledge functionality. In contrast, the protocol in [4] is constructed with zero-knowledge subproofs of membership, where these subproofs are assumed to fulfill the standard definition of zero-knowledge. Likewise, they prove that Protocol 4 is a zero-knowledge proof of knowledge, again by the standard definition and not in terms of an ideal functionality. Finally, [4] consider any efficient adversary, whereas here we limit ourselves to *non-abusing* adversaries (which suffices for our purpose). Another presentational difference is that they separately prove that part 1 is a commit-with-extract scheme and then show that this suffices for obtaining a zero-knowledge proof of knowledge. In contrast, we prove the entire protocol as one unit. For motivational discussion on the protocol and its proof of security, see [4]; we provide a technical proof here only.

In our proof below, we consider the case that P_1 is corrupted. The case that P_2 is corrupted is exactly the same. Note that the adversary $\mathcal{A}$ that controls P_1 in Π sends three types of messages: real messages belonging to Π', real messages belonging to Protocol 4 and ideal messages for the trusted party computing the zero-knowledge proof of membership functionality.

We now describe the adversary $\mathcal{A}'$ for Π'. $\mathcal{A}'$ works in the ZK-hybrid model and therefore interacts with a trusted party who computes the zero-knowledge proof of knowledge functionality. That is, $\mathcal{A}'$ sends two types of messages: real messages belonging to Π' and ideal messages for its trusted party. $\mathcal{A}'$ internally incorporates $\mathcal{A}$ and forwards all of the Π'-messages between $\mathcal{A}$ and the external party P_2. In contrast, the messages of Protocol 4 and the ideal messages that $\mathcal{A}$ sends to the trusted party for memberZK are internally dealt with by $\mathcal{A}'$. Informally speaking, in the case that $\mathcal{A}$ is proving a statement v, adversary $\mathcal{A}'$ must extract a witness w in order to send (v,w) to its trusted party (who computes the zero-knowledge proof of knowledge functionality). On the other hand, in the case that $\mathcal{A}$ is verifying a statement v, adversary $\mathcal{A}'$ must simulate an execution of Protocol 4 for $\mathcal{A}$. At the conclusion of the simulation of all Π'-messages and executions of Protocol 4, adversary $\mathcal{A}'$ outputs whatever $\mathcal{A}$ does.

As we have mentioned, all Π'-messages are forwarded by $\mathcal{A}'$ between $\mathcal{A}$ and P_2. In contrast, the messages of Protocol 4 are dealt with internally. We describe the simulation (and present the analysis) separately for the case that $\mathcal{A}$ is the prover and the case that $\mathcal{A}$ is the verifier. In the simulation described below, there is no rewinding or dependence on Π'-messages that are sent. Therefore, the simulation of Protocol 4 is not affected by the Π'-messages that $\mathcal{A}'$ forwards between $\mathcal{A}$ and the external P_2. We now describe the internal simulation of Protocol 4.

Simulating Protocol 4 when $\mathcal{A}$ is the prover. In this case, $\mathcal{A}$ provides a proof for a statement v using Protocol 4. Since $\mathcal{A}'$ works in the ZK-hybrid model it must send a pair (v, w) to its trusted party, such that if P_2 would have accepted the proof from $\mathcal{A}$ in Π then $R(v, w) = 1$, but if P_2 would have rejected the proof then $R(v, w) = 0$. (Recall that for every R it holds that $R(v, \perp) = 0$ and so $\mathcal{A}'$ can send $(v, \perp)$ to the trusted party in the case that P_2 would have rejected.) We now demonstrate how this is done.

Informally speaking, $\mathcal{A}'$ works by biasing the outcome of $r = r_1 \oplus r_2$ of the coin-tossing protocol, so that it knows $f^{-n}(r)$. In this case, it will be able compute $\tilde{r}$ and then derive $w = \tilde{c} \oplus \tilde{r}$. Note that $\mathcal{A}'$ must generate a view for $\mathcal{A}$ that is indistinguishable from its view in a real execution, in addition to extracting w. Formally, $\mathcal{A}'$ works as follows:

1. *$\mathcal{A}'$ biases the outcome of the coin-tossing protocol:*
   (a) $\mathcal{A}'$ passes $c = \mathsf{Commit}(0^n)$ to $\mathcal{A}$ (this is a commitment to "garbage").
   (b) *$\mathcal{A}$ chooses a trapdoor permutation:* $\mathcal{A}'$ receives the description of a permutation f from $\mathcal{A}$. Adversary $\mathcal{A}'$ checks that f is indeed a permutation. If not, $\mathcal{A}'$ simulates P_2 aborting and sends $(v, \perp)$ to the trusted party (causing the real, external P_2 to abort).
   (c) $\mathcal{A}'$ obtains a string r_2 from $\mathcal{A}$.
   (d) $\mathcal{A}'$ chooses $s \in_R \{0,1\}^n$, computes $r = f^n(s)$ and passes $\mathcal{A}$ the string $r_1 = r \oplus r_2$. (Notice that r_1 is chosen irrespective of the initial commitment c, and that $f^{-n}(r_1 \oplus r_2) = f^{-n}(r) = s$.)
   (e) $\mathcal{A}'$ passes $\mathcal{A}$ the pair $((c, r_1), 1)$ as if it was sent by the trusted party for the zero-knowledge proof of membership functionality.
2. *$\mathcal{A}$ sends the actual commitment:*
   $\mathcal{A}'$ receives a string $\tilde{c}$ from $\mathcal{A}$.
3. *Witness extraction:*
   $\mathcal{A}'$ computes $\tilde{r}$ and derives $w = \tilde{r} \oplus \tilde{c}$. (Note that $\mathcal{A}'$ knows $f^{-n}(r)$ and so can compute $\tilde{r}$.)
4. *$\mathcal{A}$ "proves" that it committed to a valid witness:*
   (a) $\mathcal{A}'$ obtains the ideal message that $\mathcal{A}$ intends to send to its trusted party.
   (b) If the ideal messages was $(v, \tilde{c}, f, r)$, where v is the common input and $\tilde{c}$, f and r are as were generated in the above simulation of part 1 of the protocol, then $\mathcal{A}'$ sends its trusted party the pair (v, w). Otherwise, if the values in the message are incorrect or the message is concatenated with $\perp$, then $\mathcal{A}'$ sends its trusted party the pair $(v, \perp)$.

First, note that the view of $\mathcal{A}$ in the simulation by $\mathcal{A}'$ is computationally indistinguishable from its view in a real execution. In fact, the only difference between the views is that in the simulation, $\mathcal{A}'$ sends r_1 that is *not* the value committed to in c, whereas the honest P_2 would send r_1 such that $c = \mathsf{Commit}(r_1)$. The indistinguishability therefore follows directly from the hiding property of commitments. In order to show this formally, the entire concurrent execution (including the Π' executions and all the executions of Protocol 4) must be simulated. However, the machine that distinguishes commitments to 0^n from commitments to r_1 may be given the inputs to all of these Π' executions. Therefore, the simulation of Π' is straightforward. Regarding the executions of Protocol 4, a standard hybrid argument is used. Specifically, let N be the total number of executions of Protocol 4. Then, define the ith hybrid to be such that the first i executions of

Protocol 4 are simulated by $\mathcal{A}'$ following the above instructions, and the last $N-i$ executions are run following the honest P_2's instructions. The indistinguishability of the ith and $i+1^{\text{th}}$ hybrids then follows from the hiding property of a single commitment. Details are omitted.

Next, notice that the view of $\mathcal{A}$ defines a *unique* value w that is "committed to". This holds because f is a permutation. Therefore f and r define a unique $\tilde{r}$, and $\tilde{c}$ and $\tilde{r}$ define a unique w. Furthermore, this view fully defines P_2's output, both in a real execution of Π and in the simulated execution of Π' with $\mathcal{A}'$. In order to see this, let $V_{\mathcal{A}}$ be the view of $\mathcal{A}$. Then, we separately consider two cases:

1. *Case 1 – $\mathcal{A}$'s view $V_{\mathcal{A}}$ defines a valid witness w:* If in $V_{\mathcal{A}}$ the message sent by $\mathcal{A}$ to the trusted party in part 2 of the protocol equals $(v, \tilde{c}, f, r)$, where v is the common input and $(\tilde{c}, f, r)$ are as were generated in part 1, then P_2 accepts the proof (i.e., outputs $(v, 1)$) in both Π and Π'. This holds in Π because $(v, \tilde{c}, f, r) \in L$ and so the trusted party for the zero-knowledge proof of membership functionality sends $((v, \tilde{c}, f, r), 1)$ to P_2, causing it to accept. Likewise, in the simulation by $\mathcal{A}'$ in Π', adversary $\mathcal{A}'$ would send (v, w) to its trusted party, where w is a valid witness. This trusted party computes the zero-knowledge proof of knowledge functionality and so sends the message $(v, R(v, w)) = (v, 1)$ to P_2, causing it to accept.

   In contrast, if in $V_{\mathcal{A}}$, the message sent by $\mathcal{A}$ to the trusted party in part 2 does not equal $(v, \tilde{c}, f, r)$ as above, then P_2 rejects the proof (i.e., outputs $(v, 0)$) in both Π and Π'. If the message sent by $\mathcal{A}$ concludes with $\perp$, then P_2 would reject in Π because the trusted party for memberZK always sends 0 in this case. Likewise, in this case, $\mathcal{A}'$ sends $(v, \perp)$ to its trusted party and so in Π' party P_2 also rejects. On the other hand, if the message sent by $\mathcal{A}$ just contains the "wrong" values (e.g., the wrong statement or not consistent with part 1), then again P_2 always rejects.

2. *Case 2 – $\mathcal{A}$'s view $V_{\mathcal{A}}$ defines a value w' such that $R(v, w') = 0$:* In this case, P_2 always rejects in both Π and Π'. This holds in Π because the statement sent by $\mathcal{A}$ in part 2 must either be false (in which case, the trusted party for the zero-knowledge proof of membership sends 0 to P_2), or it must contain "wrong" values (in which case, P_2 always rejects). Likewise, in Π', adversary $\mathcal{A}'$ will either send $(v, \perp)$ or (v, w') to its trusted party; in both cases P_2 receives $(v, 0)$ and so rejects.

We conclude that the view of $\mathcal{A}$ in a real execution of Π is computationally indistinguishable from its view in the simulation by $\mathcal{A}'$ in Π'. Furthermore, this view fully defines the output of P_2 from Protocol 4. Therefore, the joint distribution of $\mathcal{A}$'s view and P_2's output in Π where Protocol 4 is run, is indistinguishable from the joint distribution generated by $\mathcal{A}'$ in Π'.[20]

Simulating Protocol 4 when $\mathcal{A}$ is the verifier. In this case, $\mathcal{A}'$ receives an ideal message $(v, 1)$ from its trusted party (that computes the zero-knowledge proof of knowledge functionality) and must generate a simulation of an execution of Protocol 4 for $\mathcal{A}$. This is achieved as follows: $\mathcal{A}'$ follows the instructions for the honest prover in part 1 of the protocol; with the exception that instead of committing to a valid witness w in Step 2, it commits to 0^n instead. (Recall that $\mathcal{A}'$ cannot commit to w because it doesn't know the value.) Let $\tilde{c}$, f and r be the values that result from

[20] We remark that in the analysis of this case where $\mathcal{A}$ is the prover, we do not need to assume that $\mathcal{A}$ is non-abusing; this is needed for the case that $\mathcal{A}$ is the verifier. However, note that if *any* trapdoor permutation is used (rather than just certified permutations – see Footnote 19), then the prover must prove that the function f is a permutation. Our proof would then need to assume that $\mathcal{A}$ is non-abusing, even for the case that it plays the prover.

the execution in part 1. Then, in order to simulate part 2, $\mathcal{A}'$ just hands $\mathcal{A}$ the tuple $((v, \tilde{c}, f, r), 1)$ as if it was sent from the trusted party. This completes the simulation.

Intuitively, the view of $\mathcal{A}$ in this simulation with $\mathcal{A}'$ is indistinguishable from its view in a real execution of Π with P_2. The only difference is whether the tuple $(\tilde{c}, f, r)$ defines a commitment to a real witness w or to 0^n. Thus, indistinguishability follows from the *hiding property* of this commitment. Note that part 2 of the protocol is exactly the same in the simulation and in a real execution because $\mathcal{A}'$ plays the role of the trusted party computing the zero-knowledge proof of membership functionality.

The hiding property of the commitment $\tilde{c}$ here follows from the hiding property of the non-interactive commitment scheme of [6], and the security of the coin-tossing protocol. In particular, if $r = r_1 \oplus r_2$ is uniform (or pseudorandom), then distinguishing between a commitment to $w \in \{0,1\}^n$ and a commitment to 0^n is essentially equivalent to distinguishing between $\{f^n(U_n), (b(U_n) \cdots b(f^{n-1}(U_n))) \oplus w\}$ and $\{f(U_n), (b(U_n) \cdots b(f^{n-1}(U_n)))\}$. Since b is a hard-core bit of f, it is infeasible to distinguish between these distributions in polynomial time. The hiding property therefore follows from the security of the coin-tossing protocol that ensures that $r_1 \oplus r_2$ is pseudorandom.

We provide a proof for the case that a single execution of Protocol 4 takes place. The general case follows from a standard hybrid argument. Specifically, let N be the number executions of Protocol 4. Then, the first i executions are such that the real witness w is committed to in $\tilde{c}$, and the last $N-i$ executions are such that 0^n is committed to. (Note that the simulation of the Π' messages and the other executions of Protocol 4 can be carried out because the distinguisher for the commitments may be given the inputs to all the Π' executions. For example, the distinguisher would therefore know the real witnesses w, as required.) Indistinguishability of the neighbouring hybrids then follows from the proof of a single execution.

We now prove the case of a single execution. Let $\mathcal{A}$ be a probabilistic polynomial-time verifier, and denote by $V_n^{\mathcal{A}}(v, \alpha)$ the distribution over $\mathcal{A}$'s view when it interacts with a party who when running Protocol 4 with common input v, follows the honest prover's instructions except that it sends the value $\tilde{c} = \tilde{r} \oplus \boldsymbol{\alpha}$ in Step 2. Note that in a real execution with the honest prover, $\mathcal{A}$'s view is $V_n^{\mathcal{A}}(v, w)$ where w is a valid witness for v, whereas in the simulation with $\mathcal{A}'$ its view is $V_n^{\mathcal{A}}(v, 0^n)$. (Recall that this is the only difference between a real execution and the simulation by $\mathcal{A}'$.) Then, we prove the following: for any probabilistic polynomial-time verifier $\mathcal{A}$, it holds that

$$\left\{V_n^{\mathcal{A}}(v, w)\right\}_{v \in \{0,1\}^n, w \in R(v)} \stackrel{\mathrm{c}}{\equiv} \left\{V_n^{\mathcal{A}}(v, 0^n)\right\}_{v \in \{0,1\}^n} \tag{10}$$

where $R(v)$ denotes the witness set of v (i.e., $R(v) = \{w \mid (v, w) \in R\}$. (Note that this is actually an abuse of notation because v is dependent on the inputs to Π'.)

As we have mentioned, the hiding property is derived from the hiding property of the non-interactive commitment scheme defined by

$$C_n(\alpha) \stackrel{\text{def}}{=} \langle f^n(U_n),\ (b(U_n) \cdots b(f^{n-1}(U_n))) \oplus \alpha \rangle \tag{11}$$

Loosely speaking, the only difference between the commit-with-extract part of Protocol 4 and the commitment scheme C_n, is that in Protocol 4 the value $f^n(U_n)$ is chosen jointly by both parties (and is not determined solely by the sender). We therefore reduce Eq. (10) to the hiding property of C_n. Assume by contradiction, that there exists a probabilistic polynomial-time receiver $\mathcal{A}$, a polynomial-time distinguisher D and a polynomial $p(\cdot)$ such that for infinitely many v's

$$\mathsf{adv}_v^D \stackrel{\text{def}}{=} \left|\Pr[D(V_n^{\mathcal{A}}(v, w)) = 1] - \Pr[D(V_n^{\mathcal{A}}(v, 0^n)) = 1]\right| \geq \frac{1}{p(|v|)} \tag{12}$$

This implies that for infinitely many n's, there exists a v ($|v| = n$) such that $\mathsf{adv}_v^D \geq 1/p(n)$. For simplicity, we will consider a single v for every such n. We will therefore write adv_n^D from now on, instead of adv_v^D. We will now use D and $\mathcal{A}$ to construct a distinguisher D' that contradicts the security of the commitment scheme C_n defined in Eq. (11). Intuitively, D' receives a commitment

$$C_n(\alpha) = \langle f^n(s),\ (b(s) \cdots b(f^{n-1}(s))) \oplus \alpha \rangle$$

for input (where $s \in_R \{0,1\}^n$) and works by invoking $\mathcal{A}$ and obtaining an execution of Protocol 4 in which $r_1 \oplus r_2 = f^n(s)$. Then, since D has an advantage in distinguishing $\alpha = w$ from $\alpha = 0^n$ given f, r_1, r_2 and $\tilde{c} = (b(s) \cdots b(f^{n-1}(s))) \oplus \alpha$, this translates to D' having an advantage in distinguishing $\alpha = w$ from $\alpha = 0^n$ given $C_n(\alpha)$. We first informally describe how D' obtains an execution with $\mathcal{A}$ where $r_1 \oplus r_2 = f^n(s)$. This is achieved using the following strategy. D' invokes $\mathcal{A}$ and obtains a commitment $c = \mathsf{Commit}(r_1)$ from $\mathcal{A}$. D' then attempts to learn r_1 by running the continuation of the coin tossing protocol with $\mathcal{A}$. If it succeeds, then it rewinds $\mathcal{A}$ and sends $r_2 = f^n(s) \oplus r_1$, where $f^n(s)$ is part of its input commitment $C_n(\alpha)$. The result of the coin-tossing protocol is thus $f^n(s)$, as desired. We now proceed with the formal proof.

Before defining D', we prove the following fact:

Fact A.2 *Let $\epsilon_n^{\mathcal{A}}$ denote the probability that $\mathcal{A}$ does* not *abort during a real execution with the honest prover. Then, $\epsilon_n^{\mathcal{A}} \geq \mathsf{adv}_n^D$.*

Proof: The only place in which the honest prover uses the witness w is in Step 2 of Part 1 of Protocol 4. Note that if $\mathcal{A}$ aborts, it must be before this step (because after this step $\mathcal{A}$ only receives messages). Therefore, in any execution where $\mathcal{A}$ aborts, the view of $\mathcal{A}$ is *independent* of the value w that is committed to in Step 2. Therefore, the advantage of D in executions where $\mathcal{A}$ aborts is exactly zero. Thus, $\mathsf{adv}_n^D \leq \epsilon_n^{\mathcal{A}}$, and the fact follows. ■

By the assumption in Eq. (12), for infinitely many n's, $\mathsf{adv}_n^D \geq 1/p(n)$. Therefore, it also holds that for all of these n's, $\epsilon_n^{\mathcal{A}} \geq 1/p(n)$. We are now ready to describe D'.

Distinguisher D' receives for input a one-way permutation f and a commitment $\hat{c} = C_n(\alpha) = \langle f^n(s),\ \tilde{c} = (b(s) \cdots b(f^{n-1}(s)) \oplus \alpha \rangle$, where $s \in_R \{0,1\}^n$. Then, D' simulates an execution for $\mathcal{A}$ as follows. All of the Π' messages are perfectly simulated using the inputs that are known to D'. In addition, the (single) execution of Protocol 4 is simulated according to the following instructions:

1. *First simulation of the coin-tossing protocol:*
   (a) D' receives a commitment $c = \mathsf{Commit}(r_1)$ from $\mathcal{A}$.
   (b) D' follows the instructions for the honest prover and uniformly chooses an enhanced trapdoor permutation g and a random string $r_2 \in_R \{0,1\}^n$. D' passes g and r_2 to $\mathcal{A}$.
   (c) D' obtains some string r_1 from $\mathcal{A}$, and a message z that $\mathcal{A}$ intended to send to the trusted party.
   (d) If $z = (c, r_1)$, then D' continues. Otherwise if $z = (c, r_1) \circ \perp$ or it doesn't contain the correct values c and r_1, then D' outputs fail and halts.
2. *Second simulation of the coin-tossing protocol:*
   (a) D' rewinds $\mathcal{A}$ to the beginning and once again receives the same commitment $c = \mathsf{Commit}(r_1)$ from $\mathcal{A}$.
   (b) D' sends $\mathcal{A}$ the permutation f and the string $r_2 = f^n(s) \oplus r_1$, where r_1 is the value obtained in Step 1c and $(f, f^n(s))$ are from its input.

(c) D' obtains some string r_1' from $\mathcal{A}$, and a message z' that $\mathcal{A}$ intended to send to the trusted party. There are three possibilities at this point:

i. If z' is not equal to (c, r_1') (either because it has different values or because it equals $(c, r_1') \circ \perp$), then D' jumps to Step 4. (This is interpreted as an abort from $\mathcal{A}$, as in a real execution.)

ii. If $z' = (c, r_1')$ but $r_1' \neq r_1$, then D' halts and outputs bad.

iii. If $z' = (c, r_1')$ and $r_1' = r_1$, then D' proceeds to Step 3 below.

3. *Simulation of the actual commitment:* D' passes the string $\tilde{c}$ (from its input commitment $\hat{c}$) to $\mathcal{A}$.

4. *Output:* At the end of the simulation, D' passes the view output by $\mathcal{A}$ to D, and outputs whatever D does. (Without loss of generality, we assume that $\mathcal{A}$ always outputs its view.)

Informally speaking, if D' does not output fail or bad, then the view generated for $\mathcal{A}$ is identical to its view in a real execution. Furthermore, this view is such that $r_1 \oplus r_2 = f^n(s)$, as desired. Of course, this is only helpful if D''s probability of outputting fail or bad is small. Now, D' can output bad if $r_1' \neq r_1$, as in Step 2(c)ii. However, intuitively this cannot happen with non-negligible probability, due to the binding property of commitments and the fact that $\mathcal{A}$ is non-abusing.[21] In addition to the above, D' can output fail if it does not receive $z = (c, r_1)$ from $\mathcal{A}$ in Step 1c. Loosely speaking, the main observation here is that by combining Fact A.2 with Eq. (12), we have that $\mathcal{A}$'s non-abort probability is at least $1/p(n)$. Therefore, with probability at least $1/p(n)$ it must be that $\mathcal{A}$ convinces D' in Step 1c (otherwise, $\mathcal{A}$ has aborted). We now formally prove a bound on D''s probability of failing.

Claim A.3 *Denote the probability that D' outputs fail upon receiving input drawn from $C_n(\alpha)$ by $\mathsf{fail}_n^{D'}(\alpha)$. Then, for every α and for every n for which $\mathsf{adv}_n^D \geq 1/p(n)$, it holds that $\mathsf{fail}_n^{D'}(\alpha) < 1 - 1/p(n)$.*

Proof: First, $\mathsf{fail}_n^{D'}(\alpha)$ is the same for every α because D' only uses the value $\tilde{c}$ after all checks enabling fail have passed. Since $\tilde{c}$ is the only value that is dependent on α, we have that D''s fail probability is independent of α.

Next, we show that for any $\alpha \in \{0,1\}^n$, $\mathsf{fail}_n^{D'}(\alpha) < 1 - 1/p(n)$. Now, machine D' outputs fail if the value z that it received in Step 1c is not equal to (c, r_1). In order to bound this probability, notices that $\mathcal{A}$'s view in the first simulation of the coin-tossing with D' is *identical* to its view in a real execution. Therefore, $\mathcal{A}$'s non-abort probability until this point in the simulation is the same as in a real execution. By Fact A.2, $\mathcal{A}$'s non-abort probability $\epsilon_n^{\mathcal{A}}$ is greater than or equal to adv_n^D. Therefore, for any n where $\mathsf{adv}_n^D \geq 1/p(n)$, $\mathcal{A}$'s non-abort probability in a real execution is at least $1/p(n)$. By the above, we have that for these n's, $\mathcal{A}$'s non-abort probability in the simulated execution with D' is also at least $1/p(n)$. D' outputs fail if $\mathcal{A}$ aborts, and so the probability that D' outputs fail is at most $1 - 1/p(n)$. ■

We now show that, conditioned on the fact that D' does not output fail, D' distinguishes commitments to w from commitments to 0^n with non-negligible probability:

Claim A.4 *If D' does not output fail, then D' distinguishes $C_n(w)$ from $C_n(0^n)$ with advantage that is negligibly close to adv_n^D. That is, for every n:*

$$\left|\Pr[D'(C_n(w)) = 1 \mid D'(C_n(w)) \neq \mathsf{fail}] - \Pr[D'(C_n(0^n)) = 1 \mid D'(C_n(0^n)) \neq \mathsf{fail}]\right| > \mathsf{adv}_n^D - \mu(n)$$

[21] In this proof, it is essential that $\mathcal{A}$ is non-abusing.

Proof: In this claim, we consider the case in which $\mathcal{A}$ behaves in such a way that D' never outputs fail. There are two possible cases. In the first case, D' outputs bad. Then, D''s advantage of distinguishing $C_n(w)$ from $C_n(0^n)$ may be 0. However, this case can occur with at most negligible probability. In the second case, D' does not output bad. Therefore, when r_1 and r_2 appear in $\mathcal{A}$'s output view (as is the case when $\mathcal{A}$ does not abort), it holds that $r_1 \oplus r_2 = f^n(s)$, where $(f, f^n(s))$ is part of D''s input commitment $C_n(\alpha)$. Furthermore, when $\alpha = w$, the distribution over $\mathcal{A}$'s final view in the simulation is identical to its view $V_n^{\mathcal{A}}(v, w)$ in a real execution with the honest prover. On the other hand, when $\alpha = 0^n$, the distribution over $\mathcal{A}$'s final view in the simulation with D' is identical to its view $V_n^{\mathcal{A}}(v, 0^n)$ in a simulated execution with $\mathcal{A}'$. We stress that the two simulations by D' of the coin-tossing are completely independent. Therefore, the conditioning over D' not outputting fail has no effect over the view of $\mathcal{A}$.

We therefore have that for both cases of $\alpha = w$ and $\alpha = 0^n$,

$$\Big|\Pr[D'(C_n(\alpha)) = 1 \mid D'(C_n(\alpha)) \neq \mathsf{fail}] - \Pr[D(V_n^{\mathcal{A}}(v, \alpha)) = 1]\Big| < \mu(n)$$

where the negligible difference of $\mu(n)$ is due to the case that D' may output bad. Therefore,

$$\big|\Pr[D'(C_n(w)) = 1 \mid D'(C_n(w)) \neq \mathsf{fail}] - \Pr[D'(C_n(0^n)) = 1 \mid D'(C_n(0^n)) \neq \mathsf{fail}]\big| \geq \mathsf{adv}_n^D - 2\mu(n)$$

This completes the proof of the claim. ■

Combining Claims A.3 and A.4, we have that for any n for which $\mathsf{adv}_n^D \geq 1/p(n)$,

$$\big|\Pr[D'(C_n(w)) = 1] - \Pr[D'(C_n(0^n)) = 1]\big| > \frac{\mathsf{adv}_n^D - \mu(n)}{p(n)}$$

By the assumption in Eq. (12), there are infinitely many n's for which $\mathsf{adv}_n^D \geq 1/p(n)$. Therefore, we conclude that there are infinitely many n's for which

$$\big|\Pr[D'(C_n(w)) = 1] - \Pr[D'(C_n(0^n)) = 1]\big| > \frac{1}{p(n)^2} - \mu(n)$$

in contradiction to the hiding property of the commitment scheme C_n. Eq. (10) therefore follows, and we obtain that the simulation by $\mathcal{A}'$ is indistinguishable. That is, the view of $\mathcal{A}$ in a real execution of Π is indistinguishable from its view in Π' with the simulation by $\mathcal{A}'$. This suffices because the view of P_2 when it plays the prover always contains the statement v only. (Recall that honest parties do not write the contents of Protocol 4 onto their Π' transcript.) Therefore, the joint distribution over the view of $\mathcal{A}$ and output of P_2 in Π is indistinguishable from in Π'.

Conclusion. As we have mentioned, all of the Π'-messages are forwarded unmodified by $\mathcal{A}'$ between $\mathcal{A}$ and P_2. In addition, the executions of Protocol 4 are simulated so that the joint distribution of $\mathcal{A}$'s view and P_2's output in all executions of Protocol 4 in Π is indistinguishable from in Π'. This holds both when $\mathcal{A}$ plays the prover and the verifier. Finally, $\mathcal{A}'$ outputs whatever $\mathcal{A}$ does. Combining all this together, we obtain that the joint distributions over $\mathcal{A}$ and P_2's output in Π is indistinguishable from the joint distribution over $\mathcal{A}'$ and P_2's output in Π'. This completes the proof. ■

Obtaining Equation (3). In order to formally prove Eq. (3), take the protocol Π in Lemma A.1 to be Protocol $\tilde{\Pi}$ from Eq. (3). Note that $\tilde{\Pi}$ is exactly of the form Π that is described in Lemma A.1. Next, notice that adversary $\tilde{\mathcal{A}}$ from Eq. (3) is *non-abusing*. This follows from the construction of $\tilde{\mathcal{A}}$, and is discussed in Footnote 16. We can therefore apply Lemma A.1 and so obtain Eq. (3).