Concurrent Zero-Knowledge*

Cynthia Dwork* Moni Naor† Amit Sahai‡

July 29, 2004

Abstract

Concurrent executions of a zero-knowledge protocol by a single prover (with one or more verifiers) may leak information and may not be zero-knowledge *in toto*. In this paper, we study the problem of maintaining zero-knowledge

We introduce the notion of an (α, β) *timing constraint*: for any two processors P_1 and P_2, if P_1 measures α elapsed time on its local clock and P_2 measures β elapsed time on its local clock, and P_2 starts *after* P_1 does, then P_2 will finish after P_1 does. We show that if the adversary is constrained by an (α, β) assumption then there exist four-round almost concurrent zero-knowledge interactive proofs and perfect concurrent zero-knowledge arguments for every language in NP. We also address the more specific problem of *Deniable Authentication*, for which we propose several particularly efficient solutions. Deniable Authentication is of independent interest, even in the sequential case; our concurrent solutions yield sequential solutions *without recourse to timing*, *i.e.*, in the standard model.

1 Introduction

In a distributed computing aggregate, a collection of physically separated processors communicate via a heterogeneous network. To date, research applications of cryptographic techniques to distributed systems have overwhelmingly concentrated on the paradigm in which the system consists of n mutually aware processors trying to cooperatively compute a function of their respective inputs (see *e.g.* [7, 52]). In distinction with this traditional paradigm, processors in an aggregate in general do not know of all the other members, nor do they generally know the topology of the network. The processors are typically *not* all trying cooperatively to compute one function or perform a specific set of tasks; in general no coordination is assumed. A prime example of an aggregate is the Internet.

Electronic interactions over an aggregate, such as economic transactions, transmission of medical data, data storage, and telecommuting, pose security risks inadequately addressed in computer

*A preliminary version of this work appeared in STOC'98.

*Microsoft Research, SVC, 1065 L'Avenida, Mountain View, CA, 94043. Most of this work was performed while the author was at the IBM Almaden Research Center. Research supported by BSF Grant 32-00032-1. E-mail: dwork@microsoft.com.

†Dept. of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. Some of this work performed while at the IBM Almaden Research Center. Research supported by BSF Grant 32-00032-1. E-mail: moni.naor@weizmann.ac.il.

‡Department of Computer Science, Princeton University, Princeton, NJ 08540. Some of this work performed while at the IBM Almaden Research Center. Part of this research was performed while the author was supported by a DOD NDSEG doctoral fellowship and partially by DARPA grant DABT63-96-C-0018. E-mail: sahai@cs.princeton.edu

science research. In particular, the issue of the security of *concurrent* executions is often ignored. In this paper we address the problem of maintaining zero-knowledge under concurrency, continuing research initiated in [28] on zero-knowledge interactions in an aggregate.

1.1 Zero-Knowledge and Concurrency: A Description of the Problem

A zero-knowledge protocol is supposed to ensure that no information is leaked during its execution. Informally, a zero-knowledge proof or argument is a protocol in which a prover P attempts to convince a probabilistic polynomial-time verifier V of an assertion, *i.e.*that a string x is in a language L. Following the framework laid out in [51, 50], we consider two probability distributions in defining zero-knowledge:

1. The interaction of P and V from V's point of view.
2. The output of a probabilistic polynomial-time machine S not interacting with anyone, called the *simulator*, on input x, when given oracle access to the verifier V.

We say a proof system or argument protocol (P, V) for a language L is *zero-knowledge* if for every input x in L and for all probabilistic polynomial-time verifiers V, the two distributions above are "alike." Intuitively, the verifier gains no knowledge by interacting with the prover except that $x \in L$, since it could have run the simulator instead. The specific variants of zero-knowledge differ by the interpretation given to "alike." The most strict interpretation, leading to *perfect zero-knowledge*, requires that the distributions be identical. A slightly relaxed interpretation, leading to *statistical zero-knowledge*, requires that the distributions have negligible statistical deviation from one another. The most common interpretation, leading to *computational zero-knowledge*, requires that samples from the two distributions be indistinguishable by any polynomial-time machine.

If the prover is polynomial-time bounded, then the interaction is called an *argument* [51], but if we allow computationally unbounded provers, it is called a proof system. These distinctions do not affect the zero-knowledge condition, but indicate the severity of the soundness condition of the interaction.

For concreteness, consider a zero-knowledge interactive proof for graph Hamiltonicity (there is nothing special about the protocol or the fact that it is for Hamiltonicity), due to Blum [10]. For this we need a *string commitment* primitive. Roughly speaking, string commitment has two phases. During the *commit* phase, a *sender* and a *receiver* interact to produce a conversation α that represents a string s. The receiver should gain no information about s. If the secrecy of s depends on the computational boundedness of the receiver, we use the notation $\alpha \in C(s)$. If the secrecy of s is information-theoretic, then we use the notation $\alpha \in K(s)$. A commitment protocol in which the secrecy is information-theoretic is sometimes called a *strong-receiver* commitment, since secrecy is preserved even against a computationally unbounded receiver.

During the *reveal* phase, the sender reveals s together with information proving that α is a commitment to s. When $\alpha \in C(s)$, there is only one possible value s that can be revealed, no matter how powerful the prover. When $\alpha \in K(s)$, since secrecy is information-theoretic, α can be opened to *any* value s'; in this case its computational boundedness ensures that the sender will be unable to find two different ways of opening the commitment.

In Blum's protocol, graph Hamiltonicity is proved by the sequential iteration of the following basic block. In the first message, the prover commits to the adjacency matrix of a graph. The commitment is bit-by-bit, that is, each bit is committed to individually.

Protocol 1 Basic Block

$P \longrightarrow V$: *C(adjacency matrix A of random permutation* $\pi(G)$*)*
$V \longrightarrow P : b \in_R \{0, 1\}$
$P \longrightarrow V$: *If* $b = 0$ *then reveal* π *and open* A
If $b = 1$ *then reveal a Hamiltonian cycle in* A

In the naïve parallelization of this basic block, the prover commits to n adjacency matrices, receives a vector of n bits, and, according to the ith bit, reveals either the ith permutation and the ith adjacency matrix, or the cycle in the the ith adjacency matrix. However, the proof that the basic block is zero-knowledge fails for the naïve parallelization. Indeed, Goldreich and Krawczyk proved there is no 3-round (black-box) zero-knowledge proof or argument for any language not in BPP that achieves negligible soundness error [47]. However, in order to parallelize the basic block it suffices for the verifier to (computationally) commit to the vector of queries in advance, as shown by Goldreich and Kahan [48]:

Protocol 2 "Generic" Parallel Hamiltonicity Protocol

1. $V \longrightarrow P : K(\overrightarrow{q})$
2. $P \longrightarrow V : C(Adj(\pi_1(G)), \ldots, Adj(\pi_n(G)))$
3. $V \longrightarrow P$: *open* $\overrightarrow{q}$
4. $P \longrightarrow V$: *reply to queries*

For the simulation, we assume that the Verifier always produces valid Step 3 responses. The protocol (and simulation) without this assumption requires greater care. The basic idea for the simulation is to figure out the queries of the verifier (revealed in Step 3), and then "rewind" the simulation and commit to graphs (in Step 2) such that we can produce good replies to the queries. Specifically, if the simulator knows a query bit will be a 0, then it can simply pick a random permutation of the graph as it is supposed to; whereas if the query bit is 1, it can commit to a complete graph and reveal a random Hamiltonian cycle when required to do so in Step 4. The simulation is written here step-by-step:

Simulation:
1. $V \longrightarrow P : K(\overrightarrow{q})$
2. $P \longrightarrow V : C(garbage)$
3. $V \longrightarrow P$: open $\overrightarrow{q}$
2'. $P \longrightarrow V$: Commit to graphs suitable for $\overrightarrow{q}$
3'=3. $V \longrightarrow P$: open $\overrightarrow{q}$
4. $P \longrightarrow V$: reply to queries

Thus, this protocol is zero-knowledge and yet can have arbitrarily low soundness error.

1.1.1 Reality Strikes – The Full Concurrency Model

Unfortunately, when Protocol 2 is run concurrently, say, by a single prover interacting with multiple verifiers, the simulation fails in a manner analogous to the failure of the simulation of the naïve parallelization of the basic block. Since there are many verifiers, and since they are not all known to the prover (or provers) before interaction with the first verifier begins, there is no algorithmic way to force all subsequent verifiers to commit to their queries before the first interaction begins. Consider the following *nested* interleaving, shown in Diagram 1 below, of n colluding verifiers $V_1, \ldots, V_n$ concurrently executing Protocol 2 with a single prover.

V_1	V_2	$\ldots$	V_n
Step 1			
Step 2			
	Step 1		
	Step 2		
		$\vdots$	
			Step 1
			Step 2
			Step 3
			Step 4
		$\vdots$	
	Step 3		
	Step 4		
Step 3			
Step 4			

Diagram 1. A troublesome interleaving.

An adversary controlling the verifiers can arrange that the Step 1 commitments to queries made by verifiers $V_{i+1}, \ldots, V_n$ can depend on messages sent by the prover in Step 2 of its interaction with V_i. The difficulty with the straightforward approach is that once the queries in the interaction with V_i are opened (in Step 3), it becomes necessary to re-simulate Step 2 of the interaction with V_i, and therefore the entire simulation of the interaction with verifiers $V_{i+1}, \ldots, V_n$ must be re-simulated (since the Step 1 messages of each of these verifiers may have changed). The most deeply nested transaction, with V_n, is simulated roughly 2^n times. Indeed, it has recently been shown that transcripts of this interleaving cannot be simulated in probabilistic polynomial time unless Hamiltonicity is in BPP (see Section 1.6). Moreover, there exist protocols that are ordinarily zero-knowledge and yet fail dramatically to be zero-knowledge in the concurrent scenario [47].

1.2 Our Approach: Adding Timing Constraints

This problem of building protocols that are simulatable in the concurrent setting is indeed quite tricky. To overcome the problem of simulation for parallelism, it was possible to make use of an initial commitment by the verifier. A natural analogue of this that can easily be made to work in the concurrent scenario is to require that all verifiers engage in some kind of initial commitment before the interleaved protocols begin. Such a solution, however, is very unsatisfying as it requires a *global* requirement across all the participants in the protocols, very much against the spirit of a distributed aggregate in which parties are not necessarily even aware of each other's existence. Instead, one would like to be able to impose *local* constraints that affect only the pair of users involved in a single protocol. We achieve this goal by formally introducing into our protocols a notion which is always present in reality: the notion of time. We employ time in our protocols by imposing reasonable local timing constraints on the parties in our protocols (and in some cases, we discuss how to do so implicitly rather than explicitly using "moderately hard" functions). Using this approach, we are able to build several interesting constant round protocols that are zero-knowledge in the concurrent setting.

In this work, we consider timing constraints of only the following two types:

1. **Delays**: One party must delay the sending of some message until *at least* some specified time β has elapsed on its local clock since some specified point earlier in the protocol. Note that no special assumptions (other than the ability for the party to keep time) are needed to implement such a constraint, although delays might be somewhat inconvenient.

2. **Time-outs**: One party, the Receiver, requires that the Sender deliver its next message before some specified time α has elapsed on the Receiver's local clock since some specified point earlier in the protocol. With each such constraint, there is an assumption that an honest Sender will be able to produce and deliver its message within the alloted time. (Thus α must be chosen with this in mind.) We remark that such time-out constraints are already found in almost any secure implementation of any cryptographic protocol.

In all our protocols, we use only the above types of timing constraints, which we believe to be quite reasonable. Furthermore, we must assume some kind of synchronization of clocks of good parties. Fortunately, since our timing constraints are so undemanding, the synchronization assumption we need is quite weak: We make the assumption that all good parties have clocks that satisfy what we call an (α, β)-*constraint* (for some $\alpha \leq \beta$):

Definition 1.1 *We say a system of honest parties satisfy the* weak synchronization assumption *corresponding to an (α, β)-constraint (for some $\alpha \leq \beta$) if for any two (possibly the same) non-faulty parties P_1 and P_2, the following is always true: Suppose P_1 measures how long it takes for α time to elapse on its local clock, while P_2 measures how long it takes for β time to elapse on its local clock. Then, if P_2 begins its measurement after P_1 does (in real time), it must be the case that P_2 finishes its measurement after P_1 does, as well.*

Remark 1.1 *We make some remarks concerning the weak synchronization assumption:*

1. *An (α, β) constraint is implied by most reasonable assumptions on the behavior of clocks in a system (e.g. the linear drift assumption).*

2. *In the special case in which $P_1 = P_2$, the constraint holds trivially. In this case, the proofs are necessarily zero-knowledge, and timing only affects completeness: if α is too small (not allowing some parties sufficient time to compute and send messages), then completeness may fail to hold.*

3. *In the general case, the (α, β)-constraint may be important for correctness of the protocol for both parties involved, i.e., zero-knowledge in concurrent zero-knowledge proofs and arguments (Protocols II and II') and unforgeability (soundness) and deniability in the Deniable Authentication protocols (Protocols 9 and 10) discussed below.*

Concurrency and timing are complicated issues; we make several remarks regarding the assumptions and limitations of our model in Section 2.7 after defining the model more precisely.

It is often possible to partially eliminate the explicit use of time from concurrent zero-knowledge arguments by employing *moderately hard functions*, possibly with *shortcut* information [29][1]. Intuitively, these functions are used in order to make sure that a certain party operating within in a time limit α may not extract secret information, but an off-line simulator having sufficient time may extract this information; in this sense they supplement the (α, β)-constraint.

1.3 Deniable Authentication

The example of concurrent proofs of Hamiltonicity, described in the previous section, provides ample motivation for the study of concurrent zero-knowledge. However, the impetus for our work came from the problem of *concurrent deniable authentication.* Deniable authentication first appears in [28], which presents an extremely simple protocol for what is there termed *public-key authentication*, a relaxation of digital signatures that permits an authenticator $\mathcal{AP}$ to authenticate messages m to a second party V, but in which the authentication needn't (and, to quote [28], "perhaps shouldn't!") be verifiable by a third party. To emphasize this last point we introduce the term *deniable authentication* and strengthen the definition in [28] by insisting on *deniability.* There are many reasonable definitions of deniability; for this informal discussion one can think of zero-knowledge.

Similar to a digital signature scheme, a deniable authentication scheme can convince V that $\mathcal{AP}$ is willing to authenticate m. However, unlike in the case of digital signatures, deniable authentication does not permit V to convince a third party that $\mathcal{AP}$ has authenticated m – there is no "paper trail" of the conversation (say, other than what could be produced by V alone, if deniability is defined as zero-knowledge). Thus, deniable authentication is incomparable with digital signatures. Deniable authentication differs from the concepts of *undeniable signature* introduced in 1989 by Chaum and Van Antwerpen [21] and *chameleon signature* introduced by Krawczyk and Rabin [63], in that deniable authentication is *not* intended for ultimate adjudication by a third party, but rather to assure V – and only V – of the validity of the message (see [42] for an excellent discussion of work on undeniable signatures). In addition to addressing the privacy needs cited in the literature on undeniable signatures, zero-knowledge public-key authentication also provides a solution to a major commercial motivation for undeniable signatures: to provide proof of authenticity of software to authorized/paying customers only. In particular, proving authenticity of the software to a pirate does not in any way help the pirate to prove authenticity of pirate copies of the software to other customers.

Another nice application of deniable authentication is in authenticating "off the record" remarks that are not for attribution. The prover's public key allows a reporter to be certain of the identity of the source while providing plausible deniability to the source.

The notion of non-malleable security for a public-key authentication scheme, defined in [28], is analogous to that of *existential unforgeability under an adaptive chosen plaintext attack* for signature schemes, defined by Goldwasser, Micali, and Rivest [56], while taking care of "person-in-the-middle" attacks. The definition can be described informally as follows (see [28] for details). Assume that an authenticator P is willing to authenticate any polynomial number of messages $m_1, m_2, \ldots$, which may be chosen adaptively by an adversary $\mathcal{A}$. We say that $\mathcal{A}$ successfully attacks the scheme if a forger C, under control of $\mathcal{A}$ and pretending to be P, succeeds in authenticating to a third

[1] A short-cut behaves similarly to a trapdoor, except that the gap between what can be computed having the short-cut and not having the short-cut is polynomial rather than the (conjectured) super-polynomial gap between what can be done having the trapdoor information and not having the trapdoor information.

party D a message $m \neq m_i$, $i = 1, 2, \ldots$. Note that the forgery may occur concurrently with valid authentications by P.

This definition describes the protection afforded to the *verifier* (receiver) against an imposter trying to impersonate the *prover* (authenticator) and falsely "authenticating" the message m. A deniable authentication protocol satisfying the above definition of non-malleable security clearly also provides some protection for the prover/authenticator; for example, even though the protocol may not be zero-knowledge, it protects any private key used in the authentication to a great extent – otherwise it would be possible to impersonate the authenticator by learning the private authentication key.

The following public-key authentication protocol appears in Section 3.5 of [28] (see also [30]). P's public key is E, chosen according to a generator of a non-malleable public key cryptosystems. Roughly speaking, a public-key cryptosystem is non-malleable if, for all polynomial time relations R (with certain trivial exceptions), seeing an encryption $E(\alpha)$ "does not help" an attacker to generate an encryption $E(\beta)$ such that $R(\alpha, \beta)$. In particular, for this application we require *non-malleability under chosen ciphertext attack in the post-processing mode*, defined formally in [28] and informally below. In all our protocols we assume that the message m to be authenticated is a common input, known to both parties. Also, if any message received is of the wrong format, then the protocol is terminated. In particular, if the message received by P in Step 1 below is not an encryption of a string with prefix m, then P terminates the protocol. The concatenation of x and y is denoted $x \circ y$.

Protocol 3 Public-Key Authentication

1. $V \longrightarrow P : \gamma \in_R E(m \circ r),\ r \in_R \{0,1\}^n$
2. $P \longrightarrow V :\ r$

Although Protocol 3 was proved in [28] (Lemma 3.6) to be secure against *existential* forgery based on the non-malleability of E, Protocol 3 is not zero-knowledge. Zero knowledge is easily achieved by having V perform a zero-knowledge proof of knowledge of r before P reveals r. (A zero-knowledge interactive proof is a *proof of knowledge* if there is a polynomial time simulator to *extract* the information for which knowledge is being proved [38].) Clearly, once the interaction is zero-knowledge it yields no "paper trail" of involvement under sequential executions by the same prover/authenticator. The modified protocol is:

Protocol 4 Sequential Zero-Knowledge Deniable Authentication

1. $V \longrightarrow P :$ *Choose* $r \in_R \{0,1\}^n$ *and send* $E(m \circ r)$
2. $P \longrightarrow V : E(r)$
3. $V \longrightarrow P : s, r,$ *where* s *is the string of random bits used for the encryption in Step 1*
4. $P \longrightarrow V :$ *open* $E(r)$

A property that we require from the encryption scheme E is that for each ciphertext c there is at most one plaintext that corresponds to it, i.e. at Step 3. there is only one r that will be accepted by P as valid[2]. Also if the message is not accepted by P, then it should not send Step 4.

Sequential zero-knowledge (deniability) follows from simple rewinding. Non-malleable security holds from the non-malleability property of the encryption function under a strong type of chosen

[2]Almost all encryption schemes satisfy this property, however there are schemes designed for *deniable encryption* [18] that do not have it.

ciphertext attack (seeing $E(x)$ does not help in finding E(a suffix of x) and given $E(m \circ r)$ does not help in finding $E(m' \circ r')$ for r' related to r). We were unable to prove that Protocol 4 remains zero-knowledge under concurrent executions, and in light of the results of [61] this seems unlikely. However, it is the next example that lead us to search for new alternatives to zero-knowledge. The protocol may not be sound, and is presented only for pedagogical reasons. This protocol requires a strong-receiver commitment scheme $K_{g_1,g_2,p}(r)$ [22] to commit to a string r. $K_{g_1,g_2,p}(x)$ is defined as follows: g_1, g_2 generate the same q-sized subgroup of $\mathbb{Z}_p^*$, where q is a large prime dividing $p-1$. Given g_1, g_2, and p, commit to x by sending $g_1^x g_2^z \bmod p$ where $z \in_R \mathbb{Z}_q$. Note that this is distributed uniformly in the subgroup generated by g_1 for all x. Decommitment is by revealing x and z. Note that if the committer knows a such that $g_2 \equiv g_1^a \bmod p$, then the "commitment" can be opened arbitrarily, so the security of the commitment relies on the hardness of finding discrete logarithms modulo p.

Protocol P: Sequential Zero-Knowledge Deniable Authentication? (may not be sound)

1. $V \longrightarrow P : E(m \circ r),\ g_1, g_2, p$
2. $P \longrightarrow V : K_{g_1,g_2,p}(r)$
3. $V \longrightarrow P : s, r$, where s is the string of random bits used for the encryption in Step 1
4. $P \longrightarrow V$: open commitment to r

Protocol P (for "Pedagogical") is readily seen to be sequential zero-knowledge[3]. Intuitively, this protocol "should be" concurrent zero-knowledge as well, since Step 2 yields no information about r *even information-theoretically*. However, again standard simulation techniques fail for the interleavings described in Diagram 1. This example – the fact that we could not prove concurrent zero-knowledge even though nothing is leaked information-theoretically – motivated us to seek an alternate definition of zero-knowledge, giving rise to the results in this paper.

1.4 Related Previous Work

In his Ph.D. thesis, Feige [37] raised the question of maintaining security under concurrent composition for zero-knowledge protocols, and formulated the alternative notion of witness-indistinguishable proofs to achieve security in this setting[4]. See Bellare, Jakobsson, and Yung [5] and Goldreich, Chapter 4 [43] for discussion of attempts to construct parallel (as opposed to concurrent) zero-knowledge protocols. In particular, Goldreich and Krawczyk [47] showed the impossibility of 3-round black-box zero-knowledge proofs for membership in any language $L \notin BPP$ and Brassard, Crépeau and Yung [15] and Goldreich and Kahan [48] obtained constant-round perfect zero-knowledge and computational zero-knowledge protocols, respectively.

The study of zero-knowledge in an aggregate was initiated by Dolev, Dwork, and Naor [28], who considered the soundness of concurrent executions of zero-knowledge proofs of knowledge: the verifier faces the possibility that the prover with which it is interacting is actually using some concurrently running second interaction as an "oracle" to help answer the verifier's queries – this is the classic chess master's problem. Thus, for example in the case of a proof of knowledge, the

[3]The protocol may not be sound because of a malleability issue: there could conceivably be some way for P to write a string α in Step 2 as a function of what was received in Step 1, so that, upon seeing the Step 3 message from V, P can successfully open α to r.

[4]A proof system of a statement having two witnesses is witness-indistinguishable if the verifier, even knowing both witnesses, cannot determine which witness is used in constructing the proof.

interaction may not actually yield a proof. The problem is the possible *malleability* of the interactive proof of knowledge, formalized and addressed in [28].

The use of timing considerations to ensure soundness and zero-knowledge is new. The only work of which we are aware that uses timing in zero-knowledge protocols is Brands and Chaum [16], in which very accurate timing is needed in order to prevent person-in-the-middle attacks by distant processors (see also Beth and Desmedt [8]). Note that timing has been suggested as a cryptanalytic tool – the best example is Kocher's timing attack [62] – so it follows that any implementation of a cryptographic protocol must be time-aware in some sense. The use of moderately hard functions was introduced by Dwork and Naor [29], and their application to time-capsules was considered by Bellare and Goldwasser [4]. The notion of "time-lock puzzles" is discussed by Rivest, Shamir, and Wagner [68].

1.5 Summary of Results

We introduce timing in order to construct constant-round protocols that are black-box zero knowledge against many colluding verifiers acting in concurrent interactions. Thus, timing can be used to circumvent lower bounds for black-box concurrent zero knowledge (see Section 1.6).

To obtain our first result (Protocol 5, Section 3.1) we add timing constraints to the standard parallelization of the generic protocol given above. Although we are not able to prove that this achieves our ultimate goal, we show that this yields a constant-round concurrent computational ε-knowledge interactive proof for an $\mathsf{N}P$-complete language. Here, by "concurrent ε-knowledge," we refer to a slight weakening of the simulability condition in the definition of zero knowledge, spelled out more formally in Section 2.

By shifting to the context of arguments, where the prover is also assumed to be computationally bounded to polynomial time, we are able to obtain stronger results. In this context, we obtain a six-round *perfect* concurrent zero-knowledge argument in the timing model under the Discrete Log Assumption (Protocol 6, Section 3.2.1)

We also show how to obtain black-box concurrent zero-knowledge arguments for all of $\mathsf{N}P$ under general assumptions. We obtain a four-round computational concurrent zero-knowledge argument in the timing model under the assumption of the existence of one-to-one one-way functions (Protocol 7, Section 3.3), which becomes a five-round protocol assuming only ordinary one-way functions. This protocol has the drawback of requiring a highly inefficient (but polynomial-time) commitment scheme. We produce a 12-round computational concurrent zero-knowledge argument that does not suffer from this drawback (Protocol 8).

Finally, in Section 4, we give several examples of protocols for deniable authentication, including two protocols which yield concrete and efficient solutions to the problem.

1.6 Subsequent Work

Kilian, Petrank, and Rackoff [61], extending the approach of Goldreich and Krawczyk for the parallel case [47], presented the first lower bound on concurrent zero knowledge, showing that any problem that has a 4 or 5 round concurrent (black-box) zero knowledge interactive proof or argument must be in $\mathsf{B}PP$. Thus, a large class of known zero-knowledge interactive proofs and arguments in the standard setting for problems believed to be outside of $\mathsf{B}PP$ do not remain provably zero-knowledge in the setting with concurrency. This was first improved to a lower bound of 8 rounds by Rosen [71] and then to $\tilde{\Omega}(\log n)$ rounds by Canetti, Kilian, Petrank, and Rosen [20].

Richardson and Kilian [70] presented the first (long) concurrent black-box zero-knowledge argument for an $\mathsf{N}P$-complete language in the standard model with no timing assumptions or constraints. This was improved by Kilian and Petrank [60] to a polylogarithmic number of rounds and to $\tilde{O}(\log n)$ rounds by Prabhakaran, Rosen, and Sahai [66].

Goldreich [44] showed that our Protocol 5 below (originally due to Goldreich and Kahan but modified here to use timing assumptions) is actually a zero-knowledge proof (we show only ε-knowledge).

Other work has focused on achieving constant-round protocols by imposing still other changes to the standard model. Di Crescenzo and Ostrovsky [27] presented a non-local approach using a different kind of preprocessing-based protocol, where the requirement is that all verifiers must complete all preprocessing protocols prior to any verifier beginning any further interaction.

Motivated in part by our notion of concurrent zero knowledge, Canetti, Goldreich, Goldwasser, and Micali [19] introduce a yet further strengthening of zero knowledge called *resettable* zero knowledge – where the protocol must remain zero knowledge even against a verifier that is given the ability to reset the state of the prover arbitrarily. This requirement implies concurrent zero knowledge, and also provides additional security in settings where a potentially resettable physical device, such as a smart card, may be acting as the prover. [19] give two main results. First, they show how to translate the result of Richardson and Kilian [70] to achieve resettable zero knowledge arguments for $\mathsf{N}P$ with a polynomial number of rounds, without timing or non-locality. Second, they also consider a model with a limited form of non-locality, where all verifiers publish a public key (which may be invalid) prior to the start of any interactions[5]. In this model, though making use of potentially stronger intractability assumptions (requiring subexponential hardness), they are able to exhibit constant-round resettable zero-knowledge arguments for an $\mathsf{N}P$-complete language. Their result also makes use of trapdoor commitment schemes in a manner very similar to our own.

The existence of a Trusted Center considerably simplifies the design and proof of many concurrent zero-knowledge arguments, including arguments for all of NP, *without recourse to timing.* Dwork and Sahai have designed a timing-based preprocessing to simulate the trusted center for the purposes of achieving concurrent zero-knowledge [35]. Once a particular prover and verifier have executed the preprocessing protocol, any polynomial number of subsequent executions of a rich class of protocols will be concurrent zero-knowledge. Thus, all use of timing is moved into a preprocessing phase, executed once for each (P, V) pair, whenever in real time they choose to do it.

Dwork and Naor defined and constructed *zaps*, informally, witness-indistinguishable two-round, public-coin, proof systems, in which the first-round message can be fixed once and for all [32]. Using zaps, and the *timed commitments* of Boneh and Naor [13], they constructed timing-based 3-round black-box concurrent zero-knowledge protocols. Dwork and Stockmeyer [35] constructed 2-round black-box (concurrent) zero-knowledge proof systems in two models[6]. In the timing model their result requires the (unproven) existence of a certain class of functions. In the case of an *advice-bounded* prover, their protocol enjoys unconditional security, independent of the amount of time used by the prover. The Dwork-Stockmeyer protocol was further analyzed by Dwork, Shaltiel, Smith, and Trevisan [36], who used techniques from circuit complexity to sharpen the bounds for the case of the advice-bounded prover and to use assumptions of a more standard flavor for the

[5]The actual requirement is somewhat weaker. For details, see [19].

[6]Note that Goldreich and Oren defined *auxiliary-input* zero-knowledge, the variant of zero-knowledge now used in virtually all cryptographic applications, and showed that no language outside of BPP has a 2-round auxiliary-input zero-knowledge protocol [50].

time-bounded case.

Di Crescenzo and Ostrovksy proposed another scheme for concurrent zero-knowledge with preprocessing. Their approach requires that all concurrent protocols finish their preprocessing phase before they start the actual proof phase. No further timing assumption are needed.

Two groups of researchers have proposed eliminating timing assumptions by strengthening the assumptions regarding the environment in which the parties operate. Goldreich, Goldwasser and Micali [46] proposed a model where the verifiers have public keys as well, but these public keys are simply registered. Damgård [25] suggested a model with a shared random string where the simulator can control the value of the string. Note that such a model is *not* sufficient to yield (credible) deniable authentication.

Finally, Barak [3] has recently invented an ingenious non-black-box protocol which is a public-coins constant round zero-knowledge argument system (such protocols cannot be black-box [47].) He also applied his techniques to obtain a concurrent zero-knowledge protocol with an a priori upper bound on the number of concurrent executions (the communication costs depend on this bound)[7]

The outstanding remaining open problem with respect to the concurrent setting is whether a non-black-box protocol can defeat the $\tilde{\Omega}(\log n)$ lower bound for black-box proofs [20] (without an a priori bound on the concurrency).

2 The Model and Preliminaries

In this section, we will define our model, concurrent zero knowledge, and various other notions that we will need in the rest of the paper. We begin with some preliminaries before we define the model:

2.1 Notation and basic terminology

We say that a machine M is *polynomial time* if there exists a polynomial $p(\cdot)$ such that for every input x, we have that $M(x)$ halts within $p(|x|)$ steps, where $|x|$ denotes the length of the string x.

A *randomized* or *probabilistic* machine is one that has access to an unbiased "coin" that it can flip on command. Formally, we choose an infinite string r uniformly among infinite strings of bits, and give this as an auxiliary input to the machine. Note that if we know that a machine M will only look at n random bits, then we need only supply M with a random string of length n. For a randomized machine M, we write $M(x)$ to denote the distribution of outputs produced when given x as an input. We write $M(x;r)$ to denote the particular output obtained when the machine M is supplied with random bits r. We say that a randomized machine M is *(strictly) polynomial time* (also abbreviated *PPT*) if there exists a polynomial $p(\cdot)$ such that for every input x, we have that $M(x;r)$ halts within $p(|x|)$ steps, no matter what the random bits r are. On the other hand, we say that a randomized machine M is *expected polynomial time* if there exists a polynomial $p(\cdot)$ such that for every input x, we have that the expected number steps that $M(x)$ takes is at most $p(|x|)$.

Finally, a *non-uniform* machine is one that additional access to an "advice" string, which depends only on the length of the input to the machine. In particular, a *non-uniform polynomial-time* machine is specified by a machine M together with a set of strings $\{\gamma_n\}_{n\in\mathcal{N}}$ with the property that there exists a polynomial $p(\cdot)$ such that for every input x, we have that $M(x, \gamma_{|x|})$ takes at

[7]Indeed, there was never reason to believe that the class of black-box zero-knowledge proofs captures all possible proof systems; a skeptical view appears in [33].

most $p(|x|)$ steps before halting. One similarly defines the notion of a *randomized (or probabilistic) non-uniform polynomial-time* machine.

2.2 Proofs and Arguments

We first recall the definition of an interactive proof:

Definition 2.1 *An interactive protocol (P, V) between a computationally unbounded prover P and a probabilistic polynomial-time verifier V is said to be an* interactive proof system *for a language L with* completeness error $c(n)$ *and* soundness error $s(n)$ *if*

1. *(*Completeness*): If $x \in L$, then $Pr(P, V)(x) = \texttt{accept} \geq 1 - c(|x|)$.*
2. *(*Soundness*): If $x \notin L$, then for all P^*, $Pr(P^*, V)(x) = \texttt{accept} \leq s(|x|)$.*

In this definition, we demand that the soundness guarantee hold for *all* P^*, even machines that take longer than polynomial time. This definition has advantages in its ease of application; however, for most cryptographic applications, we assume that all parties are polynomially bounded. Therefore, it makes sense to define a notion of proof where the prover is assumed to be computationally bounded as well. This leads to the notion of a computationally-sound proof, also called an argument, first introduced by Brassard, Chaum, and Crépeau [14]:

Definition 2.2 *An interactive protocol (P, V) between a probabilistic polynomial time prover P and a probabilistic polynomial-time verifier V is said to be a* computationally-sound interactive proof system*, or* argument*, for a language L with* completeness error $c(n)$ *and* soundness error $s(n)$ *if*

1. *(*Completeness*): If $x \in L$, then $Pr(P, V)(x) = \texttt{accept} \geq 1 - c(|x|)$.*
2. *(*Soundness*): If $x \notin L$, then for all nonuniform probabilistic polynomial time P^*, we have that $Pr(P^*, V)(x) = \texttt{accept} \leq s(|x|)$.*

2.3 Ordinary Zero Knowledge

We are now ready to define the ordinary (2-party) notion of zero knowledge for proof systems and arguments.

Definition 2.3 *Let (P, V) be an interactive proof system (or argument) with negligible completeness and soundness errors, for a language L.*

- *(P, V) is said to be a* statistical zero-knowledge *proof system (or argument) if for every nonuniform probabilistic polynomial time verifier V^*, there exists an expected probabilistic polynomial time simulator S and a negligible function α (called the* simulator deviation*) such that*

$$\text{For all } x \in L, \text{ we have } ||S(x) - \text{View}_{P,V^*}(x)|| \leq \alpha(|x|). \tag{1}$$

where $||A - B||$ denotes the statistical distance (or ℓ_1 distance) between the distributions A and B.

- *A* perfect zero-knowledge *proof system (or argument) is defined in the same way, except that Condition (1) is replaced by $||S(x) - \text{View}_{P,V^*}(x)|| = 0$, where S is allowed to output 'fail' with probability at most $1/2$ and $S(x)$ denotes the conditional distribution of S given that the output is not fail.*

- *A* computational zero-knowledge *proof system (or argument) replaces Condition (1) with the requirement that the ensembles of distributions* $\{S(x)\}_{x\in L}$ *and* $\{\mathrm{View}_{P,V^*}(x)\}_{x\in L}$ *are computationally indistinguishable.*

It may seem that the definition above would be very difficult to achieve – how can we construct a simulator for each of the infinitely many possible verifier strategies? The way this has been done in all known general zero-knowledge proofs is to actually exhibit one universal simulator, which when given "black box" access to any verifier strategy, is able to produce a simulation of that verifier with the prover. This notion of "black-box zero knowledge" was first formalized by Goldreich and Oren [50]. Black-box zero knowledge is also potentially important from the point of view of deniability, since the black-box simulator allows one to exhibit a concrete simulator which will provide deniability, rather than merely proving that a simulator exists. We state the definition formally below:

Definition 2.4 *Let* (P, V) *be an interactive proof system (or argument) with negligible completeness and soundness errors, for a language* L*.*

- (P, V) *is said to be a* (general) black-box statistical zero-knowledge *proof system if there exists a probabilistic, expected polynomial time (oracle machine) simulator* S *and a negligible function* α *(called the* simulator deviation*) such that for every non-uniform probabilistic polynomial time verifier* V^**, we have:*

$$\text{For all } x \in L, \text{ we have } ||S^{V^*}(x) - \mathrm{View}_{P,V^*}(x)|| \leq \alpha(|x|). \tag{2}$$

- *A* black-box perfect zero-knowledge *proof system is defined in the same way, except that Condition (2) is replaced by* $||S^{V^*}(x) - \mathrm{View}_{P,V^*}(x)|| = 0$*, where* S *is allowed to output 'fail' with probability at most* $1/2$ *and* $S(x)$ *denotes the conditional distribution of* S *given that the output is not fail.*

- *A* black-box computational zero-knowledge *proof system replaces Condition (2) with the requirement that the ensembles* $\{S^{V^*}(x)\}_{x\in L}$ *and* $\{\mathrm{View}_{P,V^*}(x)\}_{x\in L}$ *are computationally indistinguishable ensembles of distributions.*

2.4 ε-Knowledge

In some protocols we achieve a slightly relaxed notion of zero-knowledge, which we call ε-knowledge, requiring that for any polynomial time bounded adversary $\mathcal{A}$ and for any $0 < \varepsilon = o(1)$, there exists a simulator running in time polynomial in the running time of $\mathcal{A}$ and ε^{-1} that outputs simulated transcripts with a distribution ε-indistinguishable from the distribution on transcripts obtained when the prover interacts with (verifiers controlled by) $\mathcal{A}$. (By ε-indistinguishable we mean that no polynomial time observer can distinguish the two distributions with advantage better than ε.) For a discussion of related topics, see [53].

To illustrate the significance of ε-knowledge, suppose we have a simulator that runs in time $S(n, 1/\varepsilon, \mathcal{A}(n))$, where $\mathcal{A}(n)$ is the running time of an adversary $\mathcal{A}$ interacting with the prover on inputs of length n. Suppose further that there is a task whose success can be recognized (*e.g.*, solving an NP search problem or breaking a particular cryptosystem). Suppose that there is a procedure Π that takes part in an ε-knowledge proof and then solves the given task. Let $T(n)$ be the total running time of Π (including the interaction and the solving of the task) and let $P(n)$ be the probability that Π succeeds in solving the task. Under the normal definition of zero-knowledge, where the simulator runs in time $S'(n, \mathcal{A}(n))$, we have that without the interaction one can complete the task in time $S'(n, T(n))$, with success probability $P(n) - \mu(n)$, where $\mu(n)$ is negligible.

For our model, without the interaction, one can complete the task in time $S(n, 1/2P(n), T(n))$, with success probability $P(n) - P(n)/2 = P(n)/2$ (ignoring negligible terms).

Since we assume we can recognize when the task is completed successfully, the claim above follows by a contradiction argument: if the probability were smaller, then this task would be a distinguisher between simulated and real interactions with better than $P(n)/2$ distinguishing power, a contradiction.

This implies in particular that if the original breaking task could be achieved in polynomial-time with an inverse polynomial probability of success after interacting in the ε-knowledge proof, then it will still be achievable in this way without the interaction (although possibly requiring much more time).

2.5 Weak Synchronization Assumption

We recall the weak synchronization assumption we will need in most of our protocols:

Definition 2.5 *We say a system of honest parties satisfy the* weak synchronization assumption *given by the (α, β)-constraint (for some $\alpha \leq \beta$) if for any two (possibly the same) non-faulty parties P_1 and P_2, the following is always true: Suppose P_1 measures how long it takes for α time to elapse on its local clock, while P_2 measures how long it takes for β time to elapse on its local clock. Then, if P_2 begins its measurement after P_1 does (in real time), it must be the case that P_2 finishes its measurement after P_1 does, as well.*

2.6 Our Model and Concurrent Zero Knowledge

We now consider what it means for an interactive proof system or argument to be zero knowledge against many colluding verifiers acting concurrently. We need a way of modeling the actions of many colluding verifiers. We do so by considering the worst case – that all verifiers are under the direct control of a central controlling adversary. In fact, we even give the adversary the power to create and destroy provers and verifiers at will, as well as total adaptive control over the timings of all messages exchanged between parties.

Definition 2.6 *Let (P, V) be an interactive protocol. Let the adversary A be a non-uniform probabilistic polynomial time machine which has a time bound of the polynomial $t(n)$. The* concurrent interaction *of P and A on common input x (with private inputs p_P for P and p_V for A, sometimes omitted) is a random process which proceeds as follows:*

1. *Choose random strings r_P and r_A uniformly among infinite strings over $\{0, 1\}$.*
2. *For $i = 1, 2, \ldots, t(|x|)$[8] do:*
   (a) *Let* request $= A(p_A, x, m_1, \ldots, m_{i-1}; r_A)$.

[8]Here, we are establishing a convention that the number of rounds of communication that A can engage in is bounded by its running time.

(b) *If* request $=$ (receive, P_a, V_b, m, τ)*, this indicates that prover P_a receives from verifier V_b the message m at prover P_a's local time τ. If the adversary has never output P_a before, this indicates that the new prover P_a is being created. Similarly, if the adversary has never output V_b before, this indicates that the new verifier V_b is being created. If V_b is sending its first message to P_a, then a new "copy" of the prover P is created (in our experiment) to interact with V_b. This copy of the prover is unaware of what other communication P_a might be engaging in. In our experiment, we keep track of the progress of the interactions of every pair of interacting verifiers and provers. Note that the adversary must output a local time τ which is later than all previous local times output for P_a. If this is not the case, the adversary's message is ignored. Let $m_i =$* request.

(c) *If* request $=$ (send, P_a, V_b, τ)*, this indicates that prover P_a must send its next message to V_b at the prover's local time τ. (Again, if P_a or V_b have not appeared before, then a new prover or verifier is created in our experiment.) In this case, we obtain the message m that P_a would send next to V_b in its interaction. Note that it could be that P_a would send no message – either because it is not P_a's turn in the interaction, or because P_a refuses to do so because of some rule in the protocol. Again, the adversary must output a local time τ which is later than all previous local times output for P_a. Let $m_i =$* (request, m).

(d) *If* request $=$ `halt`*, then stop.*

Furthermore, we denote by $(P \leftrightarrow A)(x)$ the random variable which represents the view of A in its interaction with P. Namely, this variable consists of the private input p_A to A together with its random coins r_A, along with all m_i.

For an adversary A to satisfy a weak synchronization (α, β)-constraint, it must always output local times in an order that respects the constraint. For example, such an adversary could *not* output the following sequence of requests:

1. (receive, $P_1, V_1, m, \tau = 0$)
2. (receive, $P_2, V_2, m, \tau = 0$)
3. (send, $P_2, V_2, \tau = \beta$)
4. (send, $P_1, V_1, \tau = \alpha$)

This would violate the constraint, since it implies that local time β elapsed on P_2's clock while less than α local time elapsed on P_1's clock.

We are now ready to define concurrent zero-knowledge proofs and arguments. Note that we adopt the black-box notion.

Definition 2.7 *Let (P, V) be an interactive proof system (or argument) with negligible completeness and soundness errors, for a language L.*

- *(P, V) is said to be a* concurrent statistical zero-knowledge *proof system (or argument) if there exists a probabilistic, expected polynomial time* [9] *(oracle machine) simulator S and a negligible function α (called the* simulator deviation*) such that for every non-uniform probabilistic*

[9]Here we note that we allow the simulation to be expected polynomial time. Unlike for standard 2-party statistical zero-knowledge proofs, we do not know if the definitions with expected and strict polynomial-time simulators are equivalent.

polynomial time adversary A,

$$\textit{For all } x \in L, \textit{ we have } ||S^A(x) - (P \leftrightarrow A)(x)|| \leq \alpha(|x|). \tag{3}$$

- *A* concurrent perfect zero-knowledge *proof system (or argument) is defined in the same way, except that Condition (3) is replaced by $||S^A(x) - (P \leftrightarrow A)(x)|| = 0$.*

- *A* concurrent computational zero-knowledge *proof system (or argument) replaces Condition (3) with the requirement that the ensembles $\{S^A(x)\}_{x\in L}$ and $(P \leftrightarrow A)(x)\}_{x\in L}$ are computationally indistinguishable ensembles of distributions.*

Note that here, for an oracle machine S to be probabilistic polynomial time (resp. probabilistic, expected polynomial time), we mean that for every non-uniform probabilistic polynomial time machine A, there exists a polynomial $t(\cdot)$ such that for all x, we have that $S^A(x)$ halts in time $t(|x|)$ (resp. expected time $t(|x|)$). The point here is that the polynomial time bound for S can depend on the polynomial bound of the machine A that S makes oracle queries to. This is necessary for the manner in which we have defined the adversary's interactions, since the number of total interactions that the adversary engages in depends on the adversary's running time. Therefore, the running time of S^A must depend on the running time of A.

We also define a notion of *concurrent ε-knowledge*, a weakening of the definition of zero knowledge:

Definition 2.8 *Let (P, V) be an interactive proof system (or argument) with negligible completeness and soundness errors, for a language L.*

- *(P, V) is said to be a* concurrent statistical ε-knowledge *proof system (or argument) if for every polynomial p, there exists a probabilistic polynomial time* [10] *(oracle machine) simulator S such that for every non-uniform probabilistic polynomial time adversary A,*

$$\textit{For all sufficiently long } x \in L, \textit{ we have } ||S^A(x) - (P \leftrightarrow A)(x)|| \leq 1/p(|x|). \tag{4}$$

- *A* concurrent computational ε-knowledge *proof system (or argument) if for every polynomial p, there exists a probabilistic polynomial time (oracle machine) simulator S such that for every non-uniform probabilistic polynomial time adversary A, for every non-uniform distinguisher D, for all sufficiently long $x \in L$, we have*

$$\left| PrD\left(S^A(x)\right) = 1 - PrD\left((P \leftrightarrow A)(x)\right) = 1 \right| \leq 1/p(|x|) \tag{5}$$

2.7 Remarks On Our Model

In this section, we make a series of remarks concerning our model.

- **"Rewinding" and Concurrency.** In the definition of black-box zero knowledge, we assume that the simulator has oracle access to the adversary, which allows it to reset the state of the adversary (which has been called "rewinding the verifier" in the literature) and control the randomness used by the adversary. In a distributed setting with concurrency, however, this

[10] For concurrent ε-knowledge, we can always restrict to strict polynomial time since the running time can depend on the threshold polynomial p.

assumption that the state of the adversary can be reset to earlier states may be difficult to interpret. For example, consider an adversary that makes use of a concurrent interaction with a third party to select its responses. In this situation, while it seems reasonable to expect that the simulator can reset the adversary machine, it may not be reasonable to expect that the simulator can reset the state of the interacting third party. We make several observations here:

1. In some sense, difficulty here is unavoidable, since the adversary could indeed be obtaining knowledge from the third party, depending on the protocol being carried out by the third party and the adversary. If the third party is engaging in the concurrent zero knowledge protocol as a prover and is assumed to be honest, then it fits in the model as another prover.
2. On the other hand, we may consider the entire complex of parties that are concurrently interacting with the provers as being one large probabilistic polynomial-time system (assuming there are only a polynomial number of concurrently interacting parties). Since this entire system can be taken together as a machine whose state can be reset, proving that a protocol is concurrent zero knowledge implies that it "leaks no knowledge to the outside world", since it is zero knowledge with respect to the entire complex of external parties.
3. This issue does cause problems in situations where black-box simulability is used to establish deniability – that is, where one tries to argue that because one can generate simulated transcripts of the protocol, one should not trust transcripts of purported executions of the protocol as evidence that the protocol took place. If however, a user uses a concurrent interaction with a trusted third party to select its responses in the protocol that is supposed to be deniable, then the argument of deniability breaks down, since the user cannot reset the state of the third party as may be needed to produce a simulation. This explains why deniability is impossible in the *common random string* model [25, 17].

- **Timing As Information.** In our adversary model, we give the adversary total adaptive control over all local clocks in the system, subject only to a weak synchronization constraint. This choice allows us to guarantee security against the widest possible array of interleavings, which an adversary could potentially force upon the system[11]. However, because we give control of the prover's message timings to the adversary, our model fails to capture the possibility that the timing of a prover's message itself could yield potentially damaging information to the adversary. Indeed, the problem of using timing information to attack systems has been discussed in the work of Kocher [62], along with methods to minimize the risk of information loss through timing information. However, because of the various engineering issues involved in protecting against timing-based knowledge, an investigation of this important issue is beyond the scope of this paper. For more information, see [62].

- **Clocks.** Another byproduct of the adversary's total control over local clocks is the implicit assumption that local clocks are indeed *local* – that a system's measurement of time is determined as a result of local computations, as opposed to external inputs. As discussed above, the need for resettability in simulation gives rise to difficulties if there are interactions with

[11] For example, an adversary may find ways to manipulate the network or loads on the prover's machine in order to manipulate timings.

third parties. For example, if the clock of a system is provided by signed messages by a trusted third party, this will cause problems in simulation, since the third party possibly cannot be reset to earlier times. This situation is ruled out in our model since we assume that the local clock times are dictated by the adversary machine. Note that in our model, the simulator does not have direct control over clocks in the system, but only indirect control through the ability to reset the state of the adversary, which controls the clocks.

- **Soundness.** Concurrency issues do not arise in proving soundness, since we may consider all interacting parties together as one probabilistic polynomial time machine (here we assume that there are at most a polynomial number of parties interacting that can effect any single prover). Thus, if we prove that no probabilistic polynomial time machine acting as prover can prove a false assertion (which is the standard 2-party definition of soundness for computationally-sound proofs), this implies that no false assertions can be proved even in a distributed environment with concurrency. We do *not* deal with issues of impersonation or misrepresenting one's knowledge. In the context of zero-knowledge proofs, such issues have been addressed in the work of Dolev, Dwork, and Naor [28], and Sahai [72].

- **Completeness.** The addition of timing constraints adds some potential threats to the completeness guarantee in protocols. In particular, a time-out may occur due to no fault of the party required to produce a timely message, say, because the network is slow. This introduces some unavoidable non-locality, and also potentially creates a vulnerability to certain kinds of denial of service attacks, where an adversary slows down the network to deny service. Nevertheless, time-outs have proven useful in real life, and are widely used. One must take care in selecting the time limits to maintain reasonable expectations on the computation of the parties involved as well as the status of the network.

- **Timing Constraints and Composition of Protocols.** While the timing constraints that we propose (delays and time limits) allow for considerable flexibility, one must nevertheless be careful when designing protocols to ensure that the timing constraints remain reasonable. When building protocols out of many components that use timing constraints, one must ensure that all the timing constraints taken together do not conflict with each other.

2.8 Commitment Schemes

Commitment schemes are a crucial ingredient in many of our protocols. They are used to enable a party to commit itself to a value while keeping the value secret. Later on, a commitment may be "opened" to reveal the value – and it should be guaranteed that a given commitment can only be "opened" to a single value.

In other words, loosely speaking, a bit commitment scheme is a protocol between two parties – a sender S and a receiver R – that has two phases: a "commitment phase," in which S commits to a value; and an "opening phase," where S reveals to R the value that it committed to. This protocol must enjoy two basic properties:

- *Secrecy:* After the "commitment phase," the receiver R should not gain any knowledge about the committed value, even if R tries to cheat.

- *Binding:* After a successful "commitment phase," there can be at most one value that S can successfully open the commitment as in the "opening phase," even if S tries to cheat.

We will explore many variations on the notion of a commitment scheme, but they will all fall into the general framework outlined above. We now discuss specific types of commitment schemes.

When formalizing either the secrecy or the binding properties mentioned above, we can choose to impose either a computational or statistical standard of security, much as we did when defining zero knowledge. Indeed, for simplicity, when considering a statistical security standard for secrecy, we will insist on *perfect* security. (See below for formal definitions.)

It is easy to show that no commitment protocol can exist that has statistical guarantees for both secrecy and binding, so we must consider a tradeoff between these two requirements. Thus, we consider both *computationally secret* (and statistically binding) schemes, and *statistically secret* (and computationally binding commitment schemes. We use the notation "$C(x)$" to denote a computationally secret commitment to x, and "$K(x)$" to denote a perfectly secret commitment to x. See [43] for formal definitions.

We will use two simple computationally secret bit commitment schemes. The first assumes the existence of a 1-1 one-way function f together with a hard-core predicate for it, denoted B. (We assume that these functions are secure against non-uniform probabilistic, expected polynomial time machines.)

Construction 1 *(*simple bit commitment*): Let $f : \{0,1\}^* \to \{0,1\}^*$ be a 1-1 one-way function, and $B : \{0,1\}^* \to \{0,1\}$ be a hard-core predicate for f. Then we consider the following construction of a computationally secret bit commitment scheme:*

1. *The initialization phase is trivial.*
2. *$C(b) = (f(s), B(s) \oplus b)$ where s is chosen uniformly from $\{0,1\}^k$. Note s denotes the random coins used by C.*
3. *$O(s, b) = s$. To open, the committer simply reveals its coins.*
4. *$V(c = (x, \tau), b, o) = 1$ iff $f(o) = x$ and $B(o) \oplus b = \tau$. The verification simply checks to see if the commitment was created according to the rules of the protocol.*

This construction in fact achieves *perfect* binding, since by the injectivity of f, no commitment string can be a commitment to both 0 and 1.

We also make use of another construction, due to Naor [64], which can be based on any one-way function. From the results of [57], we know that one-way functions imply the existence of pseudo-random generators. We use such generators in the construction:

Construction 2 *(bit commitment from any one-way function): Let $G : \{0,1\}^* \to \{0,1\}^*$ be a pseudo-random generator where $|G(s)| = 3 \cdot |s|$ for all $s \in \{0,1\}^*$. Then consider the following construction of a CSSB bit commitment scheme:*

1. *In the initialization phase, R_I simply uniformly selects $r \in \{0,1\}^{3k}$ and sends it to S_I. The private outputs of both parties is this random string r.*
2. *$C(r, b)$ chooses $s \in \{0,1\}^k$, and outputs $G(s)$ if $b = 0$ and $G(s) \oplus r$ if $b = 1$. Note that the random coins of C are this seed s.*
3. *$O(r, s, b) = s$. To open the commitment, the committer reveals his seed s.*
4. *$V(r, c, b, o)$ accepts if $b = 0$ and $G(o) = c$ or if $b = 1$ and $G(o) \oplus r = c$).*

Note that, in this construction, there is a probability of at most 2^{-k} that r might be chosen by R_I to be equal to $G(s) \oplus G(s')$ for some $s, s' \in \{0,1\}^k$. In this case, note that by choosing the commitment string $c = G(s)$, the sender could open as both a 0 (by sending the opening $o = s$) as well as a 1 (by sending the opening $o = s'$).

We note that to commit to a string, one can simply commit bit-by-bit to each bit in the string. Note however, that one needs to obtain a new random string r for each bit to be committed. This can be done initially all at once and used in subsequent commitments.

We now give an example of a *perfectly secret* string commitment scheme based on the Discrete Logarithm Problem (DLP) over prime finite fields, due to [22]. First, let us identify the specific assumption we will make:

Assumption 1 *(*Discrete Logarithm Assumption (DLA)*): Consider a generation algorithm G_{DL} which does the following: On input 1^k, it uniformly selects a prime q between 2^k+1 and 2^{k+1} (by picking random numbers and testing for primality*[12] *until it finds a prime). It then tests if $p = 2q+1$ is also prime; if not then this process is repeated until one is found. We make the number-theoretic assumption that primes of this form have sufficient density to ensure that this process will terminate in time that is expected to be polynomial in k. Note that the multiplicative group $\mathbb{Z}_p^*$ contains exactly $p-1 = 2q$ elements. Therefore it has a unique subgroup Q of size q. Random elements $g \neq 1$ are chosen in $\mathbb{Z}_p^*$ until one is found that satisfies $g^q = 1 \bmod p$ (note that this holds for at least $q-1$ choices of g). Then g must generate the subgroup Q. A random number $x \in \mathbb{Z}_q$ is chosen, and $h = g^x \bmod p$ is thus set to be a random element of Q. Finally $G_{DL}(1^k)$ outputs the tuple (p, g, h, x).*

Then we assume that for every non-uniform probabilistic, expected polynomial time machine M, there exists a negligible function $\alpha(\cdot)$ such that:

$$Pr\left[h = g^{M(p,g,h)} \bmod p\right] < \alpha(k)$$

where $(p, g, h, x) \leftarrow G_{DL}(1^k)$.

Remark 2.1 *Note that we need not insist that p have the form $2q+1$, although it is generally believed that the DLP is hardest for primes of this form. For further discussion, see [43].*

We now return to the construction of a perfectly secret string commitment scheme based on the DLP:

Construction 3 *(perfectly secret string commitment from DLP):*

1. *In the initialization phase, R_I runs G_{DL} to obtain (p, g, h, x). The values p, g, and h are sent to S_I. The private output of S_I are these values, while the private output of R_I are all four values.*

2. *$K(p, g, h, s) = g^r \cdot h^s \bmod p$, where r is chosen uniformly from $\mathbb{Z}_q$, and s is interpreted as a number between 0 and $2^k < q$. Note that r is the randomness used by K.*

3. *$O(p, g, h, r, s) = (r, s)$. To open the commitment, the committer reveals the exponent r and the string s.*

[12] For more information on Primality testing algorithms, see [2]. See also [1].

4. *The verification procedure, given the commitment c and the opening (r, s) accepts if indeed $c = g^r \cdot h^s \bmod p$.*

Since this scheme is crucial to several of our constructions, we will prove that it is indeed a perfectly secret commitment scheme, assuming the Discrete Logarithm Assumption above:

Proof. Clearly, the completeness condition is satisfied.

To see why perfect secrecy holds, notice that for any value of s, the distribution $K(p, g, h, s) = g^{(\cdot)} h^s$ is uniform over Q. So indeed, the commitment string reveals no information whatsoever about the string s being committed to.

To see why the sender is nevertheless bound computationally, consider any probabilistic polynomial time machine M which violated the binding constraint. Note that the input to M is (p, g, h) from the output of $G_{DL}(1^k)$. Suppose M outputs (c, s^0, r^0, s^1, r^1) such that $s^0 \neq s^1$, and yet $c = g^{r^0} h^{s^0} = g^{r^1} h^{s^1} \bmod p$. Now, $s^0 \neq s^1$ implies $r^0 \neq r^1 \bmod q$. Then, in fact, we have $g^{r^0 - r^1} = h^{s^1 - s^0} \bmod p$, and hence $h = g^y \bmod p$, where we compute $y = \frac{r^0 - r^1}{s^1 - s^0} \bmod q$. Thus, from M's output we can efficiently compute a $y \in \mathbb{Z}_q$ such that $h = g^y$. By the DLA, we know this can happen with at most negligible probability, which completes the proof. $\perp$

Remark 2.2 *In fact, the DLP-based construction given above has an addition "trapdoor" property that will be crucial to some of our protocols: This is the property that, if the committer learns x, then in fact it would be able to open any commitment to any desired value. Note again that the distribution of commitments is totally independent of the string s being committed to. Suppose a commitment sent was $c = g^r h^s$, where $r \in_R \mathbb{Z}_q$. Now, for any s', we know $c = g^{(r-(s'-s)x)} h^{s'}$. Furthermore, for any fixed s, s' we have that $(r - (s' - s)x) \bmod q$ is distributed uniformly in $\mathbb{Z}_q$. Thus, if the sender knows x, the sender can decommit any commitment to an arbitrarily selected value s' in a manner perfectly indistinguishable from a honest commitment to s'.*

The Feige-Shamir trapdoor commitment scheme. The central tool used in the protocol will be a trapdoor commitment scheme introduced in [40] and based on the basic block. This commitment scheme will use a computationally secret commitment scheme C as a subprocedure. The trapdoor commitment scheme will be based on a graph I (say with q nodes) that has a Hamiltonian cycle which cannot be determined in polynomial time. This graph will be chosen using a reduction from a one-way function, so that finding a Hamiltonian cycle in the graph is as difficult as inverting the one-way function (described in detail below). The main property of the commitment scheme is that one can create commitment strings that are indistinguishable from normal commitments, such that given a Hamiltonian cycle for I, one can decommit to both a 0 and a 1. The commitment scheme C_I works as follows:

Construction 4 *(Trapdoor Commitment): Let C denote the one-round computationally secret commitment scheme given in Construction 1.*

- *To commit to a 0, the sender picks a random permutation π of the nodes of I, and commits to the entries of the adjacency matrix of the permuted graph one by one, using C. To decommit, the sender reveals π and opens every entry of the adjacency matrix.*
- *To commit to a 1, the sender commits to the entries of an all-ones adjacency matrix (corresponding to the complete graph on q nodes) using C. To decommit, the sender reveals a random Hamiltonian cycle in the adjacency matrix.*

This commitment scheme certainly has the property of being computationally secret, *i.e.* the distributions $C_I(0)$ and $C_I(1)$ are computationally indistinguishable for any graph I. Note also that given a Hamiltonian cycle in I, one can open commitments to 0 as both 0 and 1. Note that the converse is also true, namely that given the opening of any commitment as both a 0 and a 1, one can extract a Hamiltonian cycle of I.

Picking a suitable graph for Construction 4: For the commitment scheme above to be binding, it must be the case that given the graph I, it is difficult to find a Hamiltonian cycle in I. To ensure that this is the case, we use a reduction from a one-way function to produce the graph. Let $f : \{0,1\}^* \to \{0,1\}^*$ be any length-preserving one-way function. Consider the following language:

$$L = \{(y_1, y_2) : \text{there exists } x \in \{0,1\}^* \text{ such that either } f(x) = y_1 \text{ or } f(x) = y_2\}$$

Clearly, $L \in \mathsf{N}P$. Because Hamiltonicity is $\mathsf{N}P$-complete,there is a polynomial-time reduction mapping instances of L to instances of Hamiltonicity that also gives a witness mapping in the other direction. More precisely, there is a polynomial-time computable function g which takes as input an instance (y_1, y_2) of L, and produces as output a graph I with the property that $I \in \text{Hamiltonicity} \iff (y_1, y_2) \in L$, together with a circuit C (called the *witness mapper*). C takes as input the description w of a cycle in I, and if w is a Hamiltonian cycle in I, it outputs an x such that either $f(x) = y_1$ or $f(x) = y_2$. Furthermore, g does not reduce the number of witnesses – if for some instance (y_1, y_2), there are two values x_1 and x_2 such that $f(x_1) = y_1$ and $f(x_2) = y_2$, then there are at least two distinct Hamiltonian cycles in G.

Thus, to choose a suitable graph for the commitment scheme, for a given n, we choose two random values[13] $x_1, x_2 \in_R \{0,1\}^n$, and let $y_1 = f(x_1)$, and $y_2 = f(x_2)$. We apply the reduction g on (y_1, y_2) to obtain a graph I on poly(n) vertices, together with the circuit C. Note again, that by the properties of g, if one finds a Hamiltonian cycle in I, using C this would yield a preimage x of either y_1 or y_2 under the one-way function f. By assumption, any non-uniform polynomial-time machine could accomplish this goal with only negligible probability over the choices of x_1 and x_2.

2.9 Deniable Authentication.

We now summarize the requirements of a deniable authentication scheme, as discussed in Section 1.3. The prover or authenticator P publishes a public key E. The verifier knows the public key and following the execution of an authentication protocol it decides whether to accept or reject the session as an authentication of a message m. Consider an adversary $\mathcal{A}$ that controls the communication of several copies of a prover P who are not aware of each other and control the verifiers with whom they interact.

The authentication protocol should satisfy:

Completeness For any message m, if the prover and verifier follow the protocol for authenticating the message m, then the verifier accepts; this can be relaxed to "accepts with high probability."

Soundness - Existential Unforgeability Consider an adversary $\mathcal{A}$ as above. Suppose that the copies of P are willing to authenticate any polynomial number of messages $m_1, m_2, \ldots$, which

[13] It may seem strange that we insist on having two possible witnesses. This is important in the proof of soundness given by Feige and Shamir [40] for the protocol we build upon. For details, see [40]. This is not important to our proof of concurrent zero knowledge.

may be chosen adaptively by the adversary $\mathcal{A}$. We say that $\mathcal{A}$ successfully attacks the scheme if a forger C, under control of $\mathcal{A}$ and pretending to be P, succeeds in authenticating to a third party D (running the verifier's V protocol) a message $m \neq m_i$, $i = 1, 2, \ldots$. The soundness requirement is that all probabilistic polynomial time $\mathcal{A}$ will succeed with negligible probability.

Zero-Knowledge - Deniability Consider an adversary $\mathcal{A}$ as above and suppose that the copies of P are willing to authenticate any polynomial number of messages. Then for each $\mathcal{A}$ there exists a polynomial-time simulator that outputs an indistinguishable transcript. Two possible relaxation are: (1) allow the simulator to depend on ε (ε -knowledge) (2) Let the simulator have access to the *real* P when it produces the simulated transcript, but P would be authenticating some fixed sequence of messages (and not the ones that were actually authenticated).

3 Concurrent Zero-Knowledge Proofs and Arguments

In this section we use timing constraints to design concurrent zero-knowledge protocols for proof and argument systems. We first present in Section 3.1 a four round "generic" protocol for NP and show that it is concurrent ε-knowledge. In Section 3.2 we present a 5-round protocol for NP, which under the hardness of the discrete log assumption is concurrent zero-knowledge. In Section 3.3 we show two more involved protocols that achieve this property but can can be based on general assumptions (existence of one-way functions)

3.1 Concurrent ε-Knowledge Proofs

We now show that adding timing constraints to Protocol 2 yields a protocol for concurrent ε-knowledge proofs of Hamiltonicity. Thus, every language in NP has a concurrent ε-knowledge proof.

Commitments in Step 2 are computational, so even an arbitrarily powerful prover is really committed. We do not require the prover to prove the same theorem in all of its concurrent interactions.

Protocol 5 Generic Concurrent ε-Knowledge Proof with Timing

1. $V \longleftrightarrow P$: *Set up statistically secret commitment scheme* K[14].
2. $V \longrightarrow P$: *Commitment* $K(q)$ *to* n *queries.*
3. $P \longrightarrow V$: $C(Adj(\pi_1(G)), \ldots, Adj(\pi_n(G)))$
4. $V \longrightarrow P$: *Open queries* q.
5. $P \longrightarrow V$: *Reply to all queries*

Timing Constraints:

[14] If the prover has a public key, then the parameters for K can appear there, obviating the need for this step.

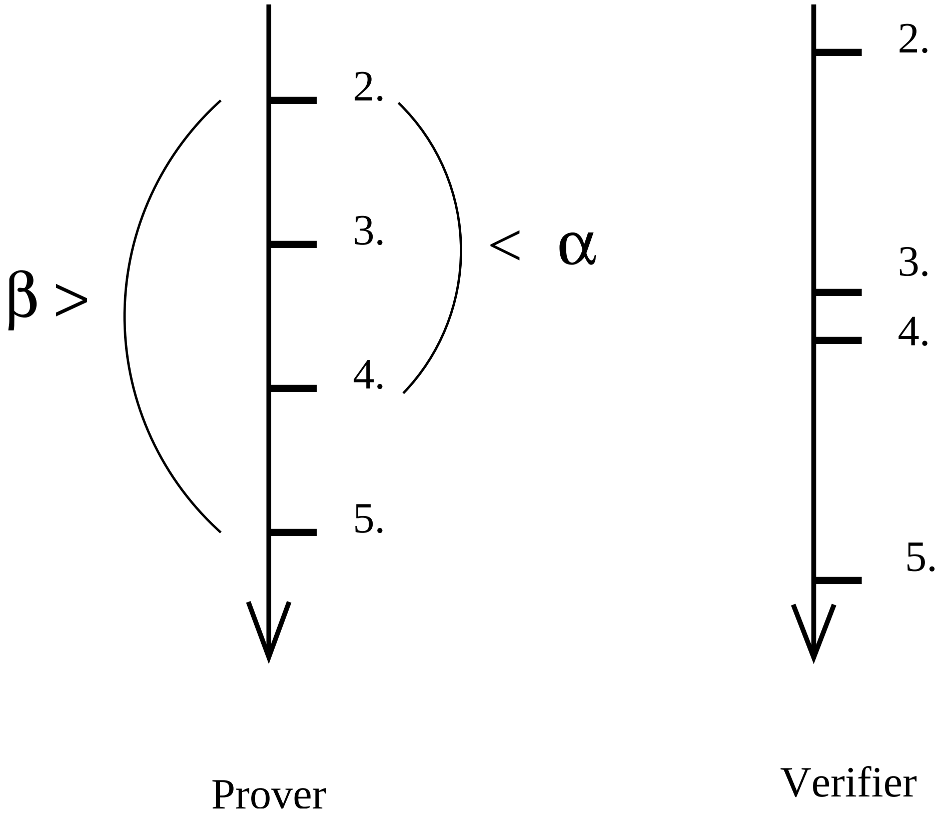

Figure 1: The timing requirements for the generic ZK protocol.

1. P requires the Step 4 message be received within α (local) time from receipt of the Step 2 message;

2. P delays Step 5 until the entire interaction, from the receipt of the Step 2 message to the sending of the Step 5 message, takes at least β local time. See Figure 1.

For the proof to be zero-knowledge we require that the adversary adhere to the (α, β)-constraint. For *completeness* we must assume that V can send the required messages in local time α.

The following theorem says that Protocol 5 is sound under concurrent executions and is concurrent ε-knowledge, *i.e.*, that there is a simulator that produces transcripts ε-indistinguishable from the true ones. Note that this is a relaxation of the usual notion of zero-knowledge. As we shall see, we remove this relaxation in Protocols 6 and 8.

Theorem 3.1 *Protocol 5 is a computational concurrent ε-knowledge proof system for graph Hamiltonicity.*

Proof. We must show completeness, soundness, and ε-knowledge. Completeness follows by inspection.

Concurrent ε-Knowledge. Recall that in the two-party case the protocol is zero-knowledge [48]. To give some intuition for the concurrent case, let us first assume that the verifier will always provide valid commitments in Step 2 and always open them in a valid and timely manner in Step 4.

We must simulate the interaction of a set of provers with many colluding verifiers acting concurrently. Difficulties may arise if, while the simulator is waiting to extract the query bits of one verifier, another verifier appears and insists on completing the entire interaction. In order to handle this, the simulator will need to extract the new verifier's query bits, *which could depend on the particular Step 3 message that the simulator produced for the first verifier.* In this case, when the simulator finishes extracting the query bits of the first verifier and then *changes* the Step 3 message sent to it, the query bits of the second verifier could change, which would mean that the entire simulation of the second verifier would have to be redone (see Diagram 1).

We solve this problem by using the timing constraints. The timing constraints of our protocol, together with the weak synchronization assumption, imply the following extremely useful *interleaving constraint* on the adversary:

Interleaving Constraint:

For any two verifiers V_1 and V_2, if V_2 provides its Step 4 message after (in real time) V_1 provides its Step 2 commitments, then it must be that the Step 5 response to V_2 will not need to be sent before V_1 provides its Step 4 message.

This is crucial to us, since any interaction can be simulated perfectly up until Step 5 – and we need the Step 4 message of the verifier to discover its queries. The simulation proceeds by simulating each interaction perfectly until it reaches Step 4, at which point it resets back and changes its Step 3 commitments. Because of the interleaving constraint, as we continue this process one verifier at a time, we will never encounter the situation where we must provide a Step 5 message for any verifier whose queries we have not already discovered and used to prepare Step 3 commitments.

This would suffice to establish full concurrent zero knowledge, were it not for the simplifying assumption we made regarding the verifiers. In reality, the verifiers controlled by the adversary need not provide Step 4 messages in a timely or correct fashion. Unfortunately, this causes a dilemma – if a verifier V refuses to cooperate at Step 4, what can we do? If we simply ignore V and move on, later we may need to reset back to an earlier time in the simulation because of another verifier, and this time V may decide to provide a timely and correct Step 4 message. Then we must reset again, and we wind up in a very similar situation to the one we started in. In later sections, we will see how to design protocols in a different manner to deal with this problem (in arguments). For this protocol, however, we can prove only concurrent ε-knowledge. This is achieved by trying to get a valid Step 4 message from each verifier repeatedly, but only a fixed (polynomial) number of times. If we succeed, then we proceed with the simulation. If the verifier never cooperates, then we assume that the verifier will *never* provide a valid Step 4 message. If the verifier later does provide such a message, we abort. Unfortunately, this can happen with inverse polynomial probability, which is why we are only able to prove ε-knowledge.

Proof Sketch of ε-Knowledge – The Simulator If $\mathcal{A}$ is $t = q(n)$-bounded, it can initiate at most $q(n)$ concurrent executions of the protocol. We first describe a procedure, whose expected running time is polynomial (in $t = q(n)$ and $1/\varepsilon$), to simulate up to t concurrent executions of the protocol. We then argue that simulated transcripts can be distinguished from actual transcripts with advantage at most ε.

Consider the first interaction, say with V_1. V_1 sends $K(q_1)$ and the simulator replies with $C(r_1)$ where r_1 is random noise. The simulator tries to extract q_1. It is willing in this process to start (at $\mathcal{A}$'s request) up to $t-1$ other concurrent interactions. In each of these concurrent interactions it sends only commitments to random noise. Because of the (α, β) constraint it never has to issue the Step 4 reply in these interactions. If, even after running up to $t-1$ concurrent interactions and waiting until time α has elapsed, the simulator has not received q_1 from V_1, then the simulator rewinds V_1 to just after Step 1 and again tries to extract q_1. If after t^4/ε trials V_1 has still not revealed q_1, then V_1 is declared *conditionally delinquent*. Intuitively, declaring a verifier *conditionally delinquent* reflects a guess by the simulator that the probability that the verifier will open its committed queries (Step 3) in a timely fashion (*i.e.*, before α has elapsed on P's clock since the receipt of the Step 1 message) given that none of the previously declared conditional delinquents opens its committed queries in a timely fashion, is at most $O(\varepsilon/t^3)$. In this case the simulator is betting that this verifier will not open its commitments (in a timely fashion) later in the simulation (thereby causing additional rewinding). Note that each of the t^4/ε trials requires $O(t\alpha)$ time, or, if we view α as a constant, then the bound is $O(t)$.

In general, after the simulator initiates the first $j-1$ interactions $E_1, E_2, \dots, E_{j-1}$, it starts E_j and tries to extract q_j. We call E_j the *current interaction*. At this point each of $E_1, \dots, E_{j-1}$ is classified into one of two categories, *extracted* or *conditionally delinquent*. *Extracted* means that the queries q_i have been extracted and a proper answer (for Step 3) prepared for V_i. This response may or may not have been sent; this depends on the scheduling controlled by $\mathcal{A}$; the important property is that V_i need not be rewound anymore. *Conditionally Delinquent* means that the verifier has been declared conditionally delinquent because it did not execute Step 4 within time α after many trials. The probability that such a verifier will open its committed queries in a timely fashion should be smaller than $\varepsilon/4t^3$.

During the current interaction E_j the extraction is done as it was done for V_1, with one change: When V_j is declared conditionally delinquent or when q_j has been extracted, E_j is rewound until after the commitment to q_j (end of Step 2). The simulation is now ready for a new current execution. This will be the next execution in which some copy of P receives the next Step 2 message. The only difference (with the extraction at E_1) is that now the simulator may have to handle conditionally delinquent V_j that wake up.

At any step in the *extraction* of q_j, if a conditionally delinquent V_i opens its queries q_i in a timely fashion, then such a trial, called a *failed* trial, is not counted among the t^4/ε attempts to extract q_j. If, however, over $3t^4/\varepsilon$ failed trials occur, then the entire simulation is aborted. We will show that the probability of this happening must be small. All the executions are rewound until the time just before Step 3 of E_j, and a different value for r_j is chosen and committed to. The rewinding puts V_i back to "sleep" – in the state in which it has committed to its queries but not yet opened them. The values q_i are now known to the simulator but they are ignored.

We treat a waking conditionally delinquent V_i differently if it awakens (within time α) while we are waiting for a new V_j following a successful extraction of q_{j-1}. This V_i should be part of the output transcript. If at such a step of the simulation (not during extraction), a conditionally delinquent V_i opens its queries q_i in a timely fashion, then essentially we abort the execution (and either restart or output the empty transcript). If this happens, we say that the *output was aborted*. As we shall see this does not occur with probability negligibly more than $\varepsilon/4t^3$.

Concurrent ε-Knowledge – Analysis of the Simulator Clearly, the simulator described above runs in time polynomial in $|x|$, t, and $1/\varepsilon$.

We now discuss the implication of a waking conditionally delinquent verifier on the closeness of the simulated distribution to the distribution on real transcripts. For any verifier V_i, when V_i commits to q_i there is an *a priori* probability, over the choices of $\mathcal{A}$ and the prover in the remainder of the t-concurrent-execution, that V_i will eventually open its commitment (Step 4). We now define a verifier V_i being *delinquent*, as opposed to conditionally delinquent, and show that verifiers declared "conditionally delinquent" are indeed delinquent.

The definition is recursive starting from V_1. Let p_i be the probability that verifier V_i opens its commitment to q_i in a timely fashion, *conditioned on the event that no delinquent verifier $V_{i'}$ ($i' < i$) wakes up within time α (from the receipt of $V_{i'}$'s Step 2 message).* If $p_i \leq \varepsilon/4t^3$ then V_i is delinquent.

If a verifier is delinquent, then in all likelihood the simulator has made the correct decision in declaring it to be conditionally delinquent. Suppose we have only two verifiers, and both are delinquent. Suppose further that in the (rare) case in which V_1 behaves "out of character" and opens its commitments, V_2 also behaves "out of character" and open its commitments in turn. The recursive definition ensures that we will not consider such cases, and the rareness of erroneously characterizing V_1 as conditionally delinquent will ensure that we do not have to.

We now compute the probability that the simulation will declare V_i conditionally delinquent if $p_i \geq \frac{\varepsilon}{4t^3}$. Recall that $\lim_{x\longrightarrow\infty}(1 - \frac{f(x)}{x})^x \sim e^{-f(x)}$ when $\frac{(f(x))^2}{x}$ is $o(1)$. Assume that $\mathcal{A}$ cannot distinguish commitments to random noise from commitments to valid permutations of the graph with advantage better than μ. If $p_i \geq \frac{\varepsilon}{4t^3}$ then the probability that none of the t^4/ε attempts to extract q_i will succeed is at most $(1-\frac{\varepsilon}{4t^3})^{t^4/\varepsilon}$. Setting $x = t^4/\varepsilon$ yields $f(x) = t/4$, so the condition on $f(x)$ is satisfied when $\varepsilon = o(1)$, and we get a probability of $e^{-t/4}$. Thus, the probability of classifying V_i as conditionally delinquent when V_i is in fact *not* delinquent, is at most $e^{-t/4} + \mu$, where μ is the probability that $\mathcal{A}$ can break the Step 3 commitment scheme and distinguish commitments of valid permutations of the graph from commitments to random noise.

The only computationally detectable difference between the transcripts produced by the simulator and the true transcripts arises from the cases that the simulation is aborted. We will show that this has a sufficiently low probability of occurring: The probability that the output is aborted is bounded by $t \cdot \frac{\varepsilon}{4t^3} \leq \varepsilon/4$. The probability that the simulation is aborted due to failed trials can be bounded as follows: Such a failure can happen in any of up to t extractions E, due to any of up to t delinquent verifiers V waking up too often. This means that V produced at least $3t^3/\varepsilon$ valid Step 4 responses out of at most $4t^4/\varepsilon$ trials. We claim, however, that if this happens, then with all but negligible probability, the probability that V produces a valid Step 4 response conditioned on the initial transcript used in extraction E is at least $1/2t$. Indeed, the multiplicative form of Chernoff's bound states that if this were not the case, the probability of observing $3t^3/\varepsilon$ valid Step 4 responses by V out of $4t^4/\varepsilon$ trials would be $e^{\Omega(t^3)}$. Since V is delinquent, however, this implies that the probability of such a transcript arising is at most $\frac{\varepsilon}{2t^2}$. Summing over all t^2 possible extractions and delinquent verifier pairs, we see that such a failure occurs with probability only negligibly more than $\varepsilon/2$. Thus the total probability of a simulation being aborted is at most negligibly more than $3\varepsilon/4$, and hence less than ε, as required.

Since the number of trials in any extraction is bounded by $4t^4/\varepsilon$, it is clear that the entire simulation runs in time polynomial in t and $1/\varepsilon$.

Soundness. Soundness is immediate from the fact that in each (P, V_i) interaction the prover prepares its Step 2 commitments with no information whatsoever about the queries. $\perp$

Remark 3.2 *In the case of arguments, the use of timing in Protocol 5 can be partially eliminated by employing moderately hard functions. For example, the verifier can use a moderately hard function for commitment in Step 2 (e.g., a weakened Fiat-Shamir function). The protocol is zero-knowledge because the simulator can compute the query from the (moderately hard to break) commitment in (large) polynomial time. For soundness the verifier would require the Step 3 message to be* *received within a reasonably short time after the Step 2 message was sent: the delay should be considerably less than the time required to break the commitment without the short-cut information. Also care should be taken to prevent the prover's commitment from having any dependency on the commitments of the verifier. This can be done by making the verifier's commitments information-theoretic.*

Note that as the theory of moderately hard functions is much less developed than the theory of "really" hard functions, such an approach is bound to be less rigorous (although by making sufficiently strong assumptions it is possible to obtain rigorous statements).

3.2 Concurrent Zero-Knowledge Arguments

In this section, we show how to overcome the difficulty we saw in the previous section by switching to the context of arguments. We present concurrent zero-knowledge arguments for for the Hamiltonicity problem. We achieve perfect zero knowledge under the Discrete Log Assumption, and computational zero knowledge assuming only one-way functions.

We were not able to prove full zero-knowledge for the protocol from the previous section because, in the simulation, P's Step 3 message had to be changed according to V's future response in Step 4. This creates a dilemma for the simulator in the case where a verifier V fails to provide a valid Step 4 message: in this case, should the simulator ignore the verifier and simply go on with the simulation for the other verifiers, or rather should the simulator "rewind" the simulation and give the verifier V another chance to respond?

- If the simulator chooses the latter option, to rewind and give V another chance to respond, then we may encounter the same situation again if V still fails to provide a valid message. How many chances should we give V? There is a clear trade-off between the running time of the simulation and how certain we can be that V will never respond. In the previous section, we chose to give V a fixed polynomial number of chances to respond with a valid message. But, as we saw, this still left an inverse polynomial probability for certain verifiers that our simulation would fail to see a valid Step 4 message even though the verifier would give a valid message later on in the simulation.

- If the simulator instead chooses the first option, to ignore the verifier V and move on, then it may happen that later, another verifier V' provides a valid Step 4 message, which forces the simulator to "rewind" the simulation in order to change the Step 3 message to V'. At this point, the original verifier V may decide to produce a valid Step 4 message! However, in this case the simulator will not be prepared to answer V's challenge. Note that this is problematic exactly because the simulator had to "travel backwards in time" to "fix" an earlier message sent to V', at which point it became susceptible again to the whims of V.

In this section, we design new protocols to avoid this problem. In these protocols, the simulator only needs to change *future* messages based on the verifier's messages, thus avoiding the problem outlined above. For instance, in one protocol, the simulator chooses P's Step 6 response (Step 6 in this protocol is analagous to Step 5 of Protocol 5 – it contains the answers to the verifier's challenges, posed in Step 5) based on the verifier's Step 5 challenge. Of course, if the simulator were able to determine the Step 6 response based on a single Step 5 message of the verifier, then a cheating prover could do the same to fool the verifier. Thus, we design the protocol such that the simulator needs to obtain two distinct Step 5 messages from the verifier in order to be able to generate a Step 6 response that the verifier will accept.

It may seem that this simulator is no better, since it has to "travel backwards in time," as well. But the crucial difference between this new simulation strategy and the one from the previous section is that the simulator only needs to "rewind" the simulation in order to extract information, rather than to *change* earlier messages in the simulation. Once the information is extracted, the simulator discards the "rewinded" simulation and returns to the original simulation.

Thus, in the simulation, if a verifier V fails to provide a valid Step 5 message, the simulator can safely ignore it and move on. If later, when dealing with another verifier V', the simulator "rewinds" the simulation and V decides to provide a valid Step 5 message, the simulator can safely disregard this message, since the simulator will eventually return to the original simulation in which V had not given a valid Step 5 message. We use timing constraints to ensure that during an extraction, the simulator will not have to provide a Step 6 response to any verifier V that provides a valid Step 5 message during the extraction procedure. As a result, the simulator can build a transcript step-by-step, never needing to go back to change some part of the transcript constructed up to that point.

In the protocols we have seen up to this point, the simulations were based on the following basic observation about the Basic Block described in Section 1.1: *if the simulator knew what the verifier's query bits would be in advance, then it would be easy to simulate the protocol.* For the approach we will take in this section, we will make use of a different, even more basic, observation about the basic block: *if the simulator could open the prover's commitments to arbitrary values, then it would be easy to simulate the protocol.* This observation is obvious but may also seem useless, since we know that commitments are supposed to be binding – one is not supposed to be able to open commitments to any value other than the one committed to. However, as we saw in Remark 2.2, there exist "trapdoor" commitment schemes, where given a trapdoor that is ordinarily kept secret, one can open commitments to arbitrary values. Such trapdoor commitment schemes are critical to the protocols of this section. Whereas in the previous section, the simulator tried to "extract" the verifier's queries, in this section, the simulators will try to "extract" the trapdoor of a commitment scheme.

3.2.1 Concurrent Perfect Zero-Knowledge Argument based on DLP

In the first protocol given below, we will use the perfectly secret (trapdoor) commitment protocol based on the discrete logarithm problem described in Construction 3, in Section 2.8.

Protocol 6 Concurrent Perfect Zero-Knowledge Argument
A graph G is the common input to both parties. The prover is assumed to know a Hamiltonian cycle w in G.

1. *The verifier does the following: Choose random prime p of the form $p = 2q + 1$ (as described in Construction 3). Choose random g of order q in $\mathbb{Z}_p^*$. Choose random $a, a' \in \mathbb{Z}_q$. Set $h = g^a \bmod p$, $h' = g^{a'} \bmod p$. Set $K = (p, g, h)$. Send K and h' to prover.*

2. *The prover picks n random permutations $\pi_1, \ldots, \pi_n$ on the nodes of G. It then sends to the verifier commitments using the K commitment scheme to all the entries of the adjacency matrices of the permuted graphs $\pi_1(G), \ldots, \pi_n(G)$. The prover also sends a commitment $K(r_P)$, where $r_P \in_R \mathbb{Z}_q$.*

3. *The verifier sends $r_V \in_R \mathbb{Z}_q$.*

4. *The prover opens its commitment to r_P (which the verifier confirms).*

5. *The verifier does the following: It sets $r = r_V + r_P \bmod q$, and sends $c := (ra + a') \bmod q$ and a vector $\overrightarrow{q} \in_R \{0,1\}^n$ of queries to prover.*

6. *In the final step, the prover checks that $h^r \cdot h' = g^c \bmod p$. If this is correct, for $i = 1, \ldots, n$:*

   - *If $\overrightarrow{q}_i = 0$, the prover opens its commitments from Step 3 to all the entries of the adjacency matrix of $\pi_i(G)$, and also sends a description of the permutation π_i. The verifier, upon receipt, verifies the correctness of the openings, and also verifies that the revealed adjacency matrix actually is the adjacency matrix of $\pi_i(G)$. If either of these conditions fail to hold, the verifier rejects the proof.*
   - *If $\overrightarrow{q}_i = 1$, the prover opens its commitments only to those entries in the adjacency matrix of $\pi_i(G)$ that correspond to edges in the Hamiltonian cycle $\pi_i(w)$. The verifier, upon receipt, verifies the correctness of the openings, and also verifies that the revealed entries of $\pi_i(G)$ do indeed form a Hamiltonian cycle. If either of these conditions fail to hold, the verifier rejects the proof.*

Timing Constraints:

1. *P requires the Step 5 message be received within α local time from receipt of the Step 1 message;*

2. *P delays the Step 6 message until at least β local time has elapsed since the receipt of the verifier's Step 5 message.*

For the zero knowledge guarantee of this protocol, we require that the adversary be constrained by the (α, β)-constraint.

Theorem 3.3 *Protocol 6 is a perfect concurrent zero-knowledge argument for Hamiltonicity, assuming the weak synchronization assumption for (α, β) and the Discrete Logarithm Assumption.*

Proof. As usual, completeness follows by inspection.

Soundness. The proof relies on the fact that, if some cheating prover P^* convinces V of a false statement with non-negligible probability, then by exploring the answers to two different query vectors (Steps 5 and 6), it is possible to obtain two different openings of a "committed" value.

Assume for the sake of contradiction that some non-uniform probabilistic polynomial time bounded cheating P^* can convince V of a false theorem with non-negligible probability ρ. Note that we can assume that P^* is deterministic without loss of generality, because we allow it to be non-uniform[15]. We argue that there exists a probabilistic polynomial time bounded M that, by interacting with P^*, can break the DLA with probability $\Omega(\rho^6)$, which is also non-negligible.

On input $p, q, g, h = g^a$, where g generates an order q subgroup of $\mathbb{Z}_p^*$, the machine M (which is trying to find a) interacts with P^* as follows.
Step 1. M chooses $c, r \in_R \mathbb{Z}_q$. M sends $K = (p, g, h)$ and $h' = g^c h^{-r}$.

[15] Thus, we can include as part of P^*'s description the "best" setting for its random coins. For more discussion and examples of this type of standard argument, see for example [43].

Step 2. P^* sends commitments using K to certain adjacency matrices, and sends a commitment $K(r_P)$ to r_P. At this point, M saves the state of P^* for later use.
Step 3. M chooses $r_V \in_R \mathbb{Z}_q$ and sends it to P^*.
Step 4. P^* opens its commitment to r_P. At this point M resets the state of P^* to just after Step 2, sets $r_V = r - r_P \bmod q$, and M re-sends its original Step 3 message, modified substituting the new value of r_V:
Step 3'. M sends $r_V = r - r_P \bmod q$. Note that since r was chosen uniformly at random, conditioned on r_P, this new value of r_V is still distributed uniformly.

Assuming that P^* responds to the Step 3' message, one of two things can happen: either P^* changes the way it opens its commitment to r_P or not. If P^* changes its opening of r_P, then from these two openings of the Step 2 commitment, M can immediately derive the discrete logarithm a.

Otherwise, the following occurs:
Step 4'. P^* opens r_P as before, such that $r = r_V + r_P \bmod q$. At this point, M saves the state of P^* for later use.
Step 5. M sends c (and note that we have arranged that $h^r \cdot h' = g^c$). and sends a query vector $\overrightarrow{q} \in_R \{0,1\}^n$.
Step 6. The check that P^* is supposed to do would succeed by definition of h'. P^* replies to the queries. M resets the state of P^* to just after Step 4'.
Step 5'. M sends c as before. Then, M chooses a new $\overrightarrow{q}' \in_R \{0,1\}^n$ and sends $\overrightarrow{q}'$ to P^*. Note that with overwhelming probability, $\overrightarrow{q}' \neq \overrightarrow{q}$.
Step 6. P^*'s check succeeds by definition of h'. P^* replies to the new queries. M now has the reply to two different queries. If both queries were answered by P^* in a manner that convinces V to accept, and yet G does not contain a Hamiltonian cycle, it must be that P^* has provided the opening of some commitment under K (made in Step 2) to two different values. From this M can extract a.

We now analyze the probability that M succeeds in obtaining the discrete logarithm a. The (P^*, V) interaction can be viewed as a tree with probability ρ of success ("success" is when P^* provides messages which causes the verifier to accept). Recall that we may assume that P^* is deterministic. Note that M's choices of p, g, h, and h' are chosen randomly but remain fixed throughout the interaction. Now, with probability at least $\rho/2$ over these choices (which are sent in Step 1), P^* will still have a probability of at least $\rho/2$ of convincing the verifier to accept[16]. Let us assume that we are in the good case, where P^* has a probability $\rho/2$ of causing the verifier to accept conditioned on these choices.

Then, with probability at least $\rho/4$ (over the choices of r_V sent in Step 3), it must be that P^* will respond to the Step 3 message of M in a way such that P^* still has a probability of $\rho/4$ of convincing the verifier to accept conditioned on its Step 3 response (which in particular implies that P^*'s Step 4 response is well-formed and valid). Thus, the probability that M chooses two Step 3 messages (once before and once after resetting the state of P^*) which lead to a state where P^* still has a $\rho/4$ chance of success is at least $(\rho/4)^2$, conditioned on being in a probability $\rho/2$ state after the choices of p, g, h, and h'.

Assume that this is the case. At this point, M may have already succeeded in extracting a. Let us assume that we are in the bad case, where M has not yet succeeded in learning a, *i.e.* that P^*'s

[16]Note that if this were not the case, it would mean that with probability $(1-\rho/2)$, there would be a probability of at most $\rho/2$ of success, while with probability $\rho/2$, there is probability at most 1. But $(1-\rho/2)\cdot(\rho/2)+(\rho/2)\cdot 1 < \rho$, which is a contradiction of our assumption that P^* has probability ρ of convincing the verifier to accept. We will use this type of argument often in this proof.

opening of r_P does not change in Step 4'. Again, note that r and c, though randomly selected, are not changed once selected. We know that with probability $\rho/8$ over these choices, P^* will have a probability at least $\rho/8$ of still convincing the verifier to accept conditioned on these choices. Thus, finally, the probability that M chooses two Step 5 messages to which P^* responds correctly is at least $(\rho/8)^2$. In this, case M succeeds in obtaining the discrete logarithm a. Thus, the overall probability of extracting a is at least

$$\frac{\rho}{2} \cdot \frac{\rho^2}{16} \cdot \frac{\rho}{8} \cdot \frac{\rho^2}{64} = \Omega(\rho^6)$$

minus a negligible probability of 2^{-n} that $\overrightarrow{q} = \overrightarrow{q}'$ during the (P^*, M) interaction.

Ordinary Zero Knowledge. Before proving concurrent zero knowledge we outline the simulation technique for proving ordinary zero knowledge in the standard model, without timing. Note that here, unlike in the protocol of the previous section, *we make no simplifying assumptions on the behavior of the verifier.* Suppose we have an arbitrary (polynomial-time) verifier V^*. The (traditional) black-box simulation for V^* works as follows.

Step 1. The simulator receives the first message from V^*, in which it sends the parameters for the commitment scheme K with trapdoor a (unknown to the simulator), along with h'. If the verifier fails to provide a message, the simulator outputs the empty transcript and terminates. The simulator saves the state S of V^* at this point for later use.

Step 2. The simulator performs the second step exactly as an honest prover would, sending commitments to permutations of the graph along with a commitment under K to r_P, chosen randomly.

Step 3. The simulator receives r_V from the verifier. If the verifier fails to provide a message, the simulator outputs the transcript so far and terminates.

Step 4. Again, the simulator behaves just as the honest prover would, revealing its commitment to r_P. Note that by the fact that the Step 2 commitment to r_P was truly independent of r_P, and the fact that r_P was chosen randomly, $r = r_V + r_P \bmod q$ is exactly uniformly distributed in $\mathbb{Z}_q$ regardless of the verifier's actions so far.

Step 5. The verifier sends $c = (r \cdot a + a') \bmod q$, where $r = r_V + r_P \bmod q$, with exactly the same conditional probability that it does in real executions (the conditioning is on the state S of the verifier after Step 1). A *valid* Step 5 message is a syntactically correct message that passes the test in Step 6, that $h^r \cdot h' = g^c$. If the verifier sends no message or sends an invalid message, then the simulator outputs the transcript (including the invalid Step 4 message, if any) and halts. If the verifier does send a valid message, then the simulator saves the state S' of the verifier and the transcript T so far, and then executes the following extraction procedure:

Extraction: Interleave the following two procedures in parallel until the first one halts.

1. Repeat until the simulator discovers the trapdoor a:
   (a) Reset the state of the verifier to just after Step 1.
   (b) Repeat the simulation strategy above, selecting a new $r'_P \in_R \mathbb{Z}_q$ each time. Note again that by the statistical secrecy property of K, regardless of the verifier's choice of r'_V in Step 3, after Step 4, $r' = r'_V + r'_P$ will be uniformly distributed in $\mathbb{Z}_q$. Thus, the probability that $r = r'$ is exactly $1/q$.

(c) If the verifier provides a valid Step 5 response (c') to a choice of $r' \neq r$, then the simulator, by solving the equations $c = ra + a'$ and $c' = r'a + a'$ (both modulo q) for a and a', can find the trapdoor a.

(d) If the verifier fails to provide a valid Step 5 response, or if $r' = r$, then the simulator repeats this process.

2. Find a by brute force, trying all q different possibilities in the equation $g^{(\cdot)} = h$.

Step 6: Now that the simulator knows the trapdoor a, it restores the verifier to state S' and continues building on the transcript T: The simulator now replies to the Step 5 queries of V^* as necessary using the trapdoor in order to perfectly simulate the distribution that the prover would provide in the protocol.

This completes the description of the simulator. By construction, we see that the simulator perfectly simulates the protocol with no error. We now compute the running time of the simulator. Let η be the probability, conditioned on the state S of the verifier after Step 1, that the verifier will send a valid Step 5 message in an execution of the protocol — that is, that the Extraction procedure is executed. Let ξ be the probability that the verifier responds with a valid Step 5 message during extraction such that $r' \not\equiv r \bmod q$.

The probability of choosing an $r' = r \bmod q$ is exactly $\frac{1}{q}$, thus $\xi \geq \eta - \frac{1}{q}$. We consider two cases:

- $\eta > \frac{2}{q}$: In this case $\eta - \frac{1}{q} > \frac{\eta}{2}$ and so $\xi \geq \frac{\eta}{2}$ and the expected number of trials during extraction is at most $\frac{2}{\eta}$. Recall that the simulation only enters the extraction phase with probability η. Thus the overall expected number of trials is at most $\eta\frac{3}{\eta} = 3$, leading to an expected $O(1)$ trials in the extractions overall.

- $\eta \leq \frac{2}{q}$: In this case the brute force search succeeds in $O(q)$ steps, for a total of an expected $O(\eta q) = O(1)$ steps.

Thus, we have that the simulation is expected polynomial time.

Concurrent Zero Knowledge We have arranged that the proof of concurrent zero knowledge is almost exactly the same as the proof for ordinary zero knowledge. We redefine a *valid* Step 5 message to specify that the message must also satisfy the timing constraint. For a verifier V_i, we redefine η to be the probability, conditioned on the state S of the adversary through the Step 1 message of V_i, that the extraction procedure will be executed. Finally, note that the timing constraints lead to an Interleaving Constraint analogous to the one from Protocol 5. This interleaving constraint ensures that during the interaction with any verifier V_i the simulator never needs to provide a Step 6 message to any V_j that sent its Step 5 message after V_i sent its Step 1 message. Thus, during the extraction of V_i, the simulator will only need to provide Step 6 answers[17] for verifiers whose trapdoor information has *already* been extracted. Thus the extraction will never get stuck because of another verifier.

The simulation is constructed message by message. If in some (P, V_i) interaction the adversary sends a valid Step 5 message for V_i, then the extraction procedure is executed to obtain V_i's trapdoor information a_i. As noted above, the timing constraints ensure that the extraction procedure can be carried out. This completes the proof of concurrent zero knowledge. $\perp$

[17] Recall that Step 6 is the only step that requires special information to simulate perfectly.

3.3 Concurrent Zero-Knowledge Under General Assumptions

In this section we present a protocol that relies only on the existence of one-way functions. We observe that the critical element of the protocol given in the previous section is the trapdoor commitment scheme, with a protocol in which the trapdoor can be extracted by the simulator. We look at the protocol of Feige and Shamir [40] (see Construction 4), and show that by introducing timing requirements and using a particular simulator, this protocol can be proven to be concurrent zero-knowledge.

We first present the protocol assuming that one-to-one one-way functions exist, in which case the protocol has four moves. We later indicate how to transform this protocol into a five-move protocol, for which we need only assume that one-way functions exist (that are not necessarily one-to-one). In the next section we will describe a commitment scheme that avoids the Cook-Levin theorem.

Protocol 7 Concurrent Computational Zero-Knowledge Argument Under General Assumptions

A graph G is the common input to both parties. The prover is assumed to know a Hamiltonian cycle w in G.

1. $V \longrightarrow P$: *Select a graph I on $q = poly(n)$ nodes containing two distinct Hamiltonian cycles w_1 and w_2, using the method described in Section 2.8 for choosing a graph in the commitment scheme of Construction 4. Let $C_I(x)$ denote commitment to x using Construction 4 with graph I. Choose $m = q^2$ random permutations $\phi_1, \dots, \phi_m$ on q elements. Send I, and entry-by-entry commitments using Construction 1 to the adjacency matrices of $\phi_1(I), \dots, \phi_m(I)$.*
2. $P \longrightarrow V$: *Prover must perform the following two actions:*
   (a) *Send $r = (r_1, \dots, r_m) \in_R \{0,1\}^m$.*
   (b) *Choose m random permutations $\psi_1, \dots, \psi_m$ on n elements. Send entry-by-entry commitments using C_I (as described above) to the adjacency matrices of $\psi_1(G), \dots, \psi_m(G)$.*
3. $V \longrightarrow P$: *Verifier must perform the following two actions:*
   (a) *For $i = 1, \dots, m$:*
      - *If $r_i = 0$, open commitments to entire adjacency matrix of $\phi_i(I)$ and reveal ϕ_i. (Prover confirms consistency.)*
      - *If $r_i = 1$, open commitments only to entries in adjacency matrix of $\phi_i(I)$ that correspond to the Hamiltonian cycle $\phi_i(w_1)$. (Prover confirms that entries revealed are all 1 and form a Hamiltonian cycle.)*

   (b) *Send $R = (R_1, \dots, R_m) \in_R \{0,1\}^m$*
4. $P \longrightarrow V$: *For $i = 1, \dots, m$:*
   - *If $R_i = 0$, open commitments (as described above) to entire adjacency matrix of $\psi_i(G)$ and reveal ψ_i. (Verifier confirms consistency.)*
   - *If $R_i = 1$, open commitments (as described above) only to entries in adjacency matrix of $\psi_i(G)$ that correspond to the Hamiltonian cycle $\psi_i(w)$. (Verifier confirms that entries revealed are all 1 and form a Hamiltonian cycle.)*

Timing Constraints*:*

1. *The Step 3 messages must be received within time α of receipt of the Step 1 message.*
2. *The Prover waits until at least time β has elapsed since receipt of the Step 3 messages before sending the Step 4 message.*

Theorem 3.4 *Protocol 7 is a concurrent computational zero-knowledge argument for Hamiltonicity, assuming the weak synchronization assumption for (α, β) and the existence of one-to-one one-way functions.*

Proof. As usual, completeness is clear. Soundness follows from the soundness of the protocol without timing constraints as proved in [40].

Ordinary Zero Knowledge. The proof we present here follows the argument given by [40], and is analogous to the proof given in the previous subsection. We will then show how the timing constraints allow the simulation to be extended to the concurrent scenario, much as in Protocol 7. Suppose we have an arbitrary (polynomial-time) verifier V^*. The (traditional) black-box simulation proceeds as follows:
Step 1. The simulator obtains the Step 1 message of the verifier, obtaining the graph I together with entry-by-entry commitments to m adjacency matrices. The simulation saves the state S of the verifier after its first message has been sent.
Step 2. For part (a), the simulator chooses $r \in_R \{0,1\}^m$, just as the real prover would. For part (b), the simulator commits (using C_I) to all 0's for each of the m adjacency matrices. Note that so far, the only difference between simulated and real interactions is that the simulator commits to all 0's, while the real prover may not.
Step 3. We define a *valid* Step 3 message to be a syntactically well-formed message such that the prover's verification checks pass for part (a). If the verifier fails to send a Step 3 message in time, or if it sends an invalid message, the simulation ends and the partial transcript so far (including the invalid Step 3 message, if any) is output. Otherwise, the simulator stores the part (a) openings of commitments to adjacency matrices (and possibly associated permutations) for r, along with R from part (b). The simulator then saves the state S' of the verifier and the transcript T so far. It then executes the following procedure in order to extract a Hamiltonian cycle in I:
Extraction: Interleave the following two procedures in parallel until the first one halts.

1. Repeat until verifier obtains a Hamiltonian cycle in I:
   (a) Reset the state of verifier to just after Step 1.
   (b) Repeat Step 2 exactly as above (picking new random $r' \in_R \{0,1\}^m$).
   (c) If verifier replies for $r \neq r'$, there is some index i such that $r_i \neq r'_i$. Thus, by examining the openings of the commitments to the i'th adjacency matrix, we obtain a Hamiltonian cycle in I, and stop.
2. Find a Hamiltonian cycle in I by brute force. Note that this procedure takes $q! = O(2^m)$ time.

There are two possibilities: First, it is possible the brute force search completes and discovers there is no Hamiltonian cycle in I. Note that this event can only occur with probability 2^{-m}, since if I has no Hamiltonian cycle, then there is only at most one value for r which could admit a valid verifier Step 3 response. In this case, the simulation aborts. The other case is that one of the two procedures uncovers a Hamiltonian cycle c in I. This allows the simulator to open its commitments from Step 2b to any desired value in a manner (computationally) indistinguishable from what occurs in actual protocols.

Step 4: The simulator resets the state of the verifier to S' (after the first valid Step 3 message was received) and adds to the transcript T as follows: For each $i = 1, \ldots, m$, the simulator picks a random permutation ψ_i on n elements. If $R_i = 0$, the simulator uses c to open the i'th adjacency matrix as $\psi_i(G)$ and reveals ψ_i. If $R_i = 1$, the simulator uses c to open a random circuit as all 1's. The simulation ends, and the transcript prepared is output.

This completes the description of the simulator. First, we observe that because $C_I(0)$ and $C_I(1)$ are computationally indistinguishable (and the simulation aborts with only negligible probability), the output of the simulation is computationally indistinguishable from transcripts from actual interactions.

We now compute the running time of the simulation. Certainly all time spent outside of the extraction procedure is polynomially bounded. We now show that the expected running time of the extraction is also polynomially bounded: Let η be the probability conditioned on state S of the verifier (after Step 1), that the verifier will send a valid Step 3 message after the simulated Step 2. (Note that we are considering *simulated* Step 2, not actual Step 2, where the commitments would differ.) Thus, conditioned on S, η is the probability that the Extraction procedure is executed. Let ξ be the probability that the verifier responds with a valid Step 3 message during extraction where $r' \neq r$. Thus, $\xi \geq \eta - 2^{-m}$. We now consider two cases:

- Case $\eta \leq 2 \cdot 2^{-m}$: In this case, since the brute force search will end in at most 2^m steps, the expected number of steps spent in the extraction procedure overall is bounded by $\eta \cdot 2^m \leq 2$.

- Case $\eta > 2 \cdot 2^{-m}$: Then, $\xi > \eta/2$. Hence, the expected number of trials during extraction is at most $\eta \cdot 1/\xi \leq 2$.

Concurrent Zero Knowledge The extension of the proof to concurrent zero knowledge is almost exactly the same as in Protocol 6. We note that again, the timing constraints ensure that during the interaction with any verifier V_i the prover never needs to provide a Step 4 message to any V_j that sent its Step 3 message after V_i sent its Step 1 message. Thus, during the extraction of V_i, the simulator will only need to provide Step 4 answers[18] for verifiers whose trapdoor information has *already* been extracted. Thus the extraction procedure will never get stuck because of another verifier.

The simulated transcript is constructed message by message. If in some (P, V_i) interaction the adversary sends a valid Step 3 message for V_i, then the extraction procedure is executed to obtain the Hamiltonian cycle for the graph that V_i sent. As noted above, the timing constraints ensure that the extraction procedure can be carried out. This completes the proof of concurrent zero knowledge. $\perp$

If we assume only that one-way functions exist, then we can substitute the basic commitment scheme C with the 2-round interactive commitment scheme of Naor [64] (this commitment scheme is described in Construction 2). This requires one additional setup round for the commitment scheme. The proof of concurrent zero-knowledge remains the same, except for one caveat: In the

[18] Recall that Step 4 is the only step that requires special information to simulate.

scheme of Naor, there is an exponentially small probability, over the setup round, that a committer could produce a commitment that can be opened both as a 0 and as a 1. If the simulator ever observes the verifier open a commitment as both a 0 and a 1, it simply aborts. Since this can only occur rarely, it does not affect the quality of the simulation.

3.4 A Construction that Avoids the Cook-Levin Theorem

To avoid going through the Cook-Levin Theorem for the trapdoor commitments, we can use the *equivocation* property of Naor's commitment scheme, observed by Di Crescenzo, Ishai, and Ostrovsky [26], and explained below. The protocol requires twelve rounds of communication. As written, this protocol requires that the prover be restricted to probabilistic polynomial time.

We first modify the commitment scheme of Naor [64] (Construction 2) so that all strings are of length $5n$ instead of length $3n$. Thus, the protocol requires a pseudo-random generator G that stretches n-bit seeds into pseudo-random strings of length $5n$.

- The BASIC COMMIT protocol:
  1. The receiver chooses a random $5n$-bit *pad* r and sends r to the sender.
  2. To commit to a bit B, the sender chooses an n-bit seed s, and sends to the receiver the $5n$-bit long string $d = d_1 d_2 \ldots d_{5n}$, where

$$d_i = \begin{cases} [G(s)]_i & \text{if } r_i = 0 \\ [G(s)]_i \oplus B & \text{if } r_i = 1 \end{cases}$$

- The REVEAL protocol: The sender sends s and B; the receiver computes $G(s)$ and verifies that d is consistent with these values.

A pair of strings s_1 and s_2 *fools* a $5n$-bit string r if

1. For all i such that $r_i = 0 :\ [G(s_1)]_i = [G(s_2)]_i$.
2. For all i such that $r_i = 1 :\ [G(s_1)]_i \neq [G(s_2)]_i$.

It follows immediately from the definition that the pair (s_1, s_2) fools r if and only if $r = G(s_1) \oplus G(s_2)$. Thus, each pair of seeds can fool at most one pad r.

Claim 3.1 *([64]) The probability, over choice of r, that there exists a pair of seeds s_1 and s_2 that fools r is at most 2^{-3n}.*

Protocol 8 is based on several techniques:

- **Equivocal Commitment**. This is a property of the strong-sender commitment scheme of [64] first suggested in [26]: If the simulator has control over the pad in the BASIC COMMIT protocol, then the "commitment" can be opened at the REVEAL stage either as a zero or as a one. Using this, the simulated prover can "commit" to an adjacency matrix and then can open the commitments as needed to carry out the simulation. The protocol is designed so that there exists a simulator that controls the pad. When we write $C_r(x)$ we mean commitment with pad r to value x. The pad used in committing the graphs will be the exclusive-or of r_1, chosen by V, and r_2, chosen by P. A similar technique was recently applied in [27].

- **Extraction of r_1.** Suppose Alice wants to commit to r_1 of length ℓ. She chooses a random p_1 of length ℓ, sets $p_2 = r_1 \oplus p_1$, and commits to p_1 and to p_2. Bob asks to see either p_1 or p_2. In the context of Protocol 8, Alice is the *verifier* and Bob is the *simulator*. If the simulator obtains both p_1 and p_2, then it can extract r_1. The simulator will use its knowledge of r_1 to control the outcome of the pad, as discussed above.

  To assure extraction with very high probability this must be be repeated many times in parallel with different pairs p_1^j, p_2^j in each execution (where always $p_1^j \oplus p_2^j = r_1$). Here, it is possible to save on communication by employing good *erasure-correcting* codes. Alice should encode the string r_1 so that it is sufficient to obtain, say, 1/4th of the indices in order to reconstruct r_1 (we don't care about errors, since they will result in aborting the protocol anyway). Now each bit is split at random into two bits (so that the Xor of the two bits is the original) and the jth commitment is to those bits. The prover gets to chose which of the two he gets. With high probability, when the extraction is done two different execution will be different in at least 1/4th of the places (even for the closest pair) and using the erasure correction procedure r_1 can be reconstructed.

- The overall idea will be the same as in Protocol 6: the prover's commitment to the graphs will be based on an equivocal commitment scheme whose pad will be chosen jointly by the verifier and prover. The simulator, by extracting the verifier's contribution to the pad, will be able to compute a value for the prover's contribution to the pad so as to allow equivocal commitment, which will allow the simulation to proceed. However, in order to ensure that the prover cannot use the equivocal commitment to cheat, we add another level of equivocal commitments: We also allow the verifier to use an equivocal commitment scheme to commit to its contribution to the prover's pad. If the verifier uses a false pad, then it can ensure that the prover's pad is almost never equivocal. We will argue that since the prover cannot distinguish false pads from random pads for the verifier, the prover cannot cheat.

In the sequel, r_1 and r_2 are strings of length $5nk$, where k is the total number of bits committed to in Step 10, and d_1, d_2, e_1, and e_2 are each strings of length $25n^2k$. Each d_i and e_i is conceptually broken into a sequence $5nk$ blocks of length $5n$. In Step 2, each of the $50n^2k$ bit commitments is performed using the same pad r. However, in Step 5, the commitment to the ith bit of p_1 (respectively, p_2) is performed using the ith consecutive block of $5n$ bits of $d_1 \oplus d_2$ (respectively $e_1 \oplus e_2$). Similarly, in Step 10 the commitment to the ith bit is done using the ith consecutive block of $5n$ bits of $r_1 \oplus r_2$.

For simplicity we describe the protocol with a single p.

Protocol 8 Concurrent Computational Zero-Knowledge Argument Under General Assumptions and w/o Cook-Levin

1. $V \longrightarrow P : r \in_R \{0,1\}^{5n}$
2. $P \longrightarrow V : C_r(d_1)$ *and* $C_r(e_1)$ *for* $d_1, e_1 \in_R \{0,1\}^{25n^2k}$
3. $V \longrightarrow P : d_2, e_2 \in_R \{0,1\}^{25n^2k}$
4. $P \longrightarrow V : d_1, e_1$. *These are* not *openings, just the sending of strings.*
5. $V \longrightarrow P$: *Choose* $r_1, p_1 \in_R \{0,1\}^{5nk}$. *Set* $p_2 = r_1 \oplus p_1$. *Send commitments* $C_{d_1 \oplus d_2}(p_1)$ *and* $C_{e_1 \oplus e_2}(p_2)$.

6. $P \longrightarrow V$: $i \in_R 1, 2$

7. $V \longrightarrow P$: *open* p_i

8. $P \longrightarrow V$: r_2

9. $V \longrightarrow P$: *Open the other commitment* p_{3-i}*;* $r_1 =_{\mathrm{def}} p_1 \oplus p_2$.

10. $P \longrightarrow V$: $C_{r_1 \oplus r_2}(Graphs)$

11. $V \longrightarrow P$: *the queries* $\overrightarrow{q}$

12. $P \longrightarrow V$: *answer the queries and give all the information to "open"* d_1 *and* e_1*, whose values were sent in Step 4. V should verify the consistency with Step 4.*

Timing Constraints:

1. *The Step 7 message must be received within time* α *of receipt of the Step 5 message.*

2. *The Prover waits until at least time* β *has elapsed since receipt of the Step 7 message before sending the Step 12 message.*

Theorem 3.5 *Protocol 8 is sound and concurrent zero knowledge.*

Proof. We argue ordinary (sequential) zero knowledge (without timing constraints). The extension to concurrent zero knowledge (with timing constraints) is straightforward using the technique from the proofs of Protocols 6 and 8.

The intuition is that the simulator will extract both p_1 and p_2 in order to compute r_1. It will then choose r_2 so that the Step 10 commitment is equivocal. This will permit the simulator to "answer" arbitrary queries.

There is one subtlety: we must ensure that a cheating verifier V^* cannot choose d_2 and e_2 in some bizarre way as a function of the commitments to d_1 and e_1, so that the commitments with pad $d_1 \oplus d_2$ or $e_1 \oplus e_2$ are equivocal. If V^* could do this, then the simulation might fail, since, for example, the value for p_{3-i} obtained in the extraction may differ from the one obtained in Step 9 in the simulation. This would result in the "wrong" value of r_1 being computed in from the extraction, and a consequent "wrong" value of r_2 being selected in Step 8. In this case the simulator's commitments in Step 10 may not be equivocal, and the simulation would fail.

It is to avoid this problem that the actual *opening* of d_1 and e_1 (as opposed to just the sending of the strings) is delayed until Step 12, as we next explain. We will use V^*'s (presumed) ability to equivocate to violate the semantic security of the commitment in Step 2.

In the sequel, we focus on d_1 and d_2. An identical argument holds for e_1 and e_2. The argument will show that V^* cannot equivocate on even one bit of p_1 (the analogous argument for the e's shows that V^* cannot equivocate on any bit of p_2). Recall that d_1 and d_2 are strings of length $25n^2k$, and that we use the ith consecutive block of $5n$ bits to commit to the ith bit of p_1. The next discussion applies to any block of length $5n$. To simplify terminology, we just let d_1, d_2, and d_1' be 5n-bit long strings. The discussion applies to each block of this length.

Let $d_1, d_1' \in_R \{0,1\}^{5n}$. We argue that with overwhelming probability over choice of d_1 and d_1', there does not exist d_2 such that both

- $d_1 \oplus d_2$ is "bad": $\exists s_1, s_2 \in \{0,1\}^n$ such that $G(s_1) \oplus G(s_2) = d_1 \oplus d_2$;

- $d_1' \oplus d_2$ is "bad": $\exists s_1', s_2' \in \{0,1\}^n$ such that $G(s_1') \oplus G(s_2') = d_1' \oplus d_2$.

Suppose for the sake of contradiction that there exists d_2 satisfying the conditions above. Then

$$g(s_1) \oplus g(s_2) \oplus g(s_1') \oplus g(s_2') = d_1' \oplus d_1 \tag{6}$$

There are only 2^{4n} choices of the four seeds, and 2^{5n} choices for $d =_{\text{def}} d_1 \oplus d_1'$, so with overwhelming probability over choice of d there do not exist seeds s_1, s_2, s_1', s_2' satisfying Equation 6.

Let S be a sender and R a receiver. R will use V^* to break the commitment scheme. First, R runs V^* and obtains its Step 1 message r. R sends r to S. S chooses one of the two messages and commits to it. Without loss of generality, we will *call* the one committed to d_1, and the other one d_1'.

R will now interact with V^*, simulating P and running the simulation through Step 9. The presumed ability of V^* to cheat on opening its commitments is something that the simulator can test for (opening something differently during extraction than during the main branch of the simulation). Thus V^* can be used as a distinguisher: its ability to equivocate when given d_1 in Step 4 will differ from its ability to equivocate when given d_1' in Step 4. This violates the semantic security of the commitment scheme.

Thus, using standard techniques the simulator can indeed extract r_1 from V^* in expected polynomial time, assuming that V^* replies with its Step 7 message with polynomial probability. As usual, if V^* does not send the Step 7 in the main branch of the simulation, then the simulation halts.

The only difference between the simulated conversations and the actual conversations with the honest prover P is that $r_1 \oplus r_2$ is truly random in the real conversations while it is pseudo-random in the simulation.

Soundness is straightforward if the prover cannot equivocate in Step 10. This is immediate if $r_1 \oplus r_2$ is truly random. The problem here is analogous to the problem solved in proving zero knowledge: we must show that a cheating P^* cannot choose r_2 so that $r_1 \oplus r_2$ permits equivocation.

Suppose for the sake of contradiction that P^* can indeed prove false theorems with non-negligible probability; colloquially, P^* can cheat. The intuition for our proof strategy is to interact with P^*, including rewinding, to extract d_1 and e_1, and then to choose d_2 and e_2 so that $d_1 \oplus d_2$ and $e_1 \oplus e_2$ permit V to equivocate in Steps 7 and 9. Once V can equivocate, it can force r_1 to be independent of r_2 by opening p_{3-i} (and hence r_1) at random in Step 9. P^* cannot equivocate in this case and so it will "get stuck" when trying to prove a false theorem (or a theorem to which it has no witness). (It is because we wish to be able to equivocate, so as to trap P^* as just described, that each bit of p_1 and p_2 is committed to independently, since otherwise the equivocation could be detected.) We will show that if P^* can cheat in real executions then it can be used to distinguish random pads $d_1 \oplus d_2$ and $e_1 \oplus e_2$ from pseudo-random pads.

We use the following well known property (see, for example, [45]). of pseudo-random sequences. *The ability to efficiently distinguish a polynomial-sized collection of of pseudo-random strings from a polynomial-sized collection of of truly random strings can be used to construct an efficient distinguisher for single pseudo-random strings from single truly random strings.* Note that in normal executions, assuming V does not cheat, $d_1 \oplus d_2$ and $e_1 \oplus e_2$ are both sequences of $5n$-bit long truly random strings.

Consider three scenarios.

A. Real (P^*, V) executions: the pads $d_1 \oplus d_2$ and $e_1 \oplus e_2$ are truly random; V opens p_1 and p_2 to the values chosen in Step 5.

B. Executions arranged by rewinding P^*, so that the pads $d_1 \oplus d_2$ and $e_1 \oplus e_2$ are pseudo-random, with d_2, e_2 chosen to permit arbitrary equivocating, but the values p_1 and p_2 that are revealed are the ones chosen in Step 5.

C. The "trapping" executions, arranged by rewinding P^* so that the pads $d_1 \oplus d_2$ and $e_1 \oplus e_2$ are pseudo-random, the value opened in Step 7 is the one chosen in Step 5, but the value opened in Step 9 is chosen at random on the fly.

The only difference between Scenarios A and B is that the pads are truly random in the former and pseudo-random in the latter. Thus, any difference in ability of P^* to cheat in these two scenarios can be used to distinguish sequences of truly random strings from sequences of pseudo-random strings.

The only difference between Scenarios B and C is that in the former the "real" value chosen in Step 5, before P^* chose r_2, is the one revealed in Step 9, while in the latter the value revealed is randomly chosen *after* r_2 is fixed. Note that the two scenarios are *identical* before r_2 is chosen, so r_2 is chosen in exactly the same way in both scenarios (however P^* wants to choose it). Thus, $r_1 \oplus r_2$ is distributed identically in these two scenarios. Since in Scenario C it is uniform, P^* cannot cheat in Scenario C because with overwhelming probability $r_1 \oplus r_2$ does not permit equivocation; thus equivocation by P^* is impossible in Scenario B and P^* cannot cheat in this scenario. By the indistinguishability of truly random and pseudo-random pads, P^* must not be able to cheat in Scenario A either. $\perp$

4 Protocols for Concurrent Deniable Authentication

The problem of achieving concurrent deniable authentication is solved by the general techniques of the Section 3. These techniques can be adapted to give deniable authentication protocols. However, a further exploration of the problem yields several "special purpose" solutions that can be implemented at a low computational cost. We describe two such protocols in this Section. The timing constraints in one of them is different than in previous protocols, since *both* parties are involved in the enforcement.

4.1 Timed Deniable Authentication

In Protocol 9 below we modify Protocol 4 by adding timing constraints. These are essentially the same constraints that were used for the generic protocol (Protocol 5), the goal being to ensure that in all executions the total time exceeds the maximum time of the first 3 steps in any concurrent execution.

The prover P has a public key E, chosen from a non-malleable cryptosystem secure against chosen ciphertext attacks in the post-processing mode. P wishes to authenticate a message m to the Verifier V.

Protocol 9 Concurrent Deniable Authentication

1. $V \longrightarrow P$: *Choose* $r \in_R \{0,1\}^n$ *and send* $E(m \circ r)$
2. $P \longrightarrow V$: $E(r)$
3. $V \longrightarrow P$: s, r, *where* s *is the string of random bits used for the encryption in Step 1*
4. $P \longrightarrow V$: *open* $E(r)$

Timing Constraints:

1. P requires the Step 3 message be received within α local time from receipt of the Step 1 message;
2. P delays Step 4 until the entire interaction, from the receipt of the Step 1 message to the sending of the Step 4 message, takes at least β local time.

This is as in Figure 1. We assume that the adversary is constrained by the (α, β) constraint.

Theorem 4.1 *Protocol 9 is sound and is concurrent ε-knowledge.*

Proof. The proof of soundness follows from the security of the encryption scheme E, that it is non-malleable against a chosen ciphertext post-processing attack, as was shown in Lemma 3.6 of [28].

The proof of concurrent ε-knowledge is essentially identical to the proof of of Theorem 3.1. $\perp$

Given the elegant and efficient non-malleable encryption scheme of Cramer and Shoup [24], based on the Decisional Diffie-Hellman assumption, Protocol 9 is also efficient.

4.2 On-Line Deniable Authentication

The next protocol does not achieve zero-knowledge; indeed there will be digitally signed evidence that a conversation has taken place. However, the protocol ensures that the signature is only on random noise and a commitment to (different) random noise. Other than the signatures on random noise, the protocol achieves deniability by having the prover eventually release what is essentially a "session key" a short while after the end of the "session." Therefore the weakened form of deniability obtained is as follows: suppose that the prover is willing to authenticate some sequence of messages $m_1, m_2, \ldots m_\ell$. Then for any verifier V' there is a simulator that communicates with the above prover and produces a transcript that is indistinguishable from the transcript obtained by V' communicating with an authenticator willing to authenticate this sequence. The above notion is satisfactory in that apart from the fact that the prover authenticated some messages, no other information is leaked and no receipt is left with the verifier. This relaxation allows us to obtain an efficient protocol for the task.

In Protocol 10 the function $auth_k$ can be any keyed authentication function or MAC, such as DES. The important property is that for a random k the adversary cannot guess with non-negligible probability $auth_k(m)$ given $auth_k(m')$ even if $m \neq m'$ are chosen by the adversary, provided k is chosen at random and not known to the adversary. The signature function *sign* must be existentially unforgeable (see [56] for definitions and [23, 31] for efficient constructions). The protocol ensures that (eventually) the session key used will be shared by P and V. Note that V cannot *check* the Step 3 authentication until it receives r_1 in Step 4.

The prover P has a public key E, chosen from an existentially unforgeable signature scheme. P wishes to authenticate a message m to the Verifier V.

Protocol 10 On-Line Deniable Authentication Using Signatures

1. $P \longrightarrow V$: *Choose* $r_1 \in_R \{0,1\}^n$ *and send* $C(r_1)$
2. $V \longrightarrow P$: *Choose* $r_1 \in_R \{0,1\}^n$ *and send* r_2
3. $P \longrightarrow V$: $auth_{r_1 \oplus r_2}(m)$
4. $P \longrightarrow V$: $sign(C(r_1) \circ r_2)$, r_1

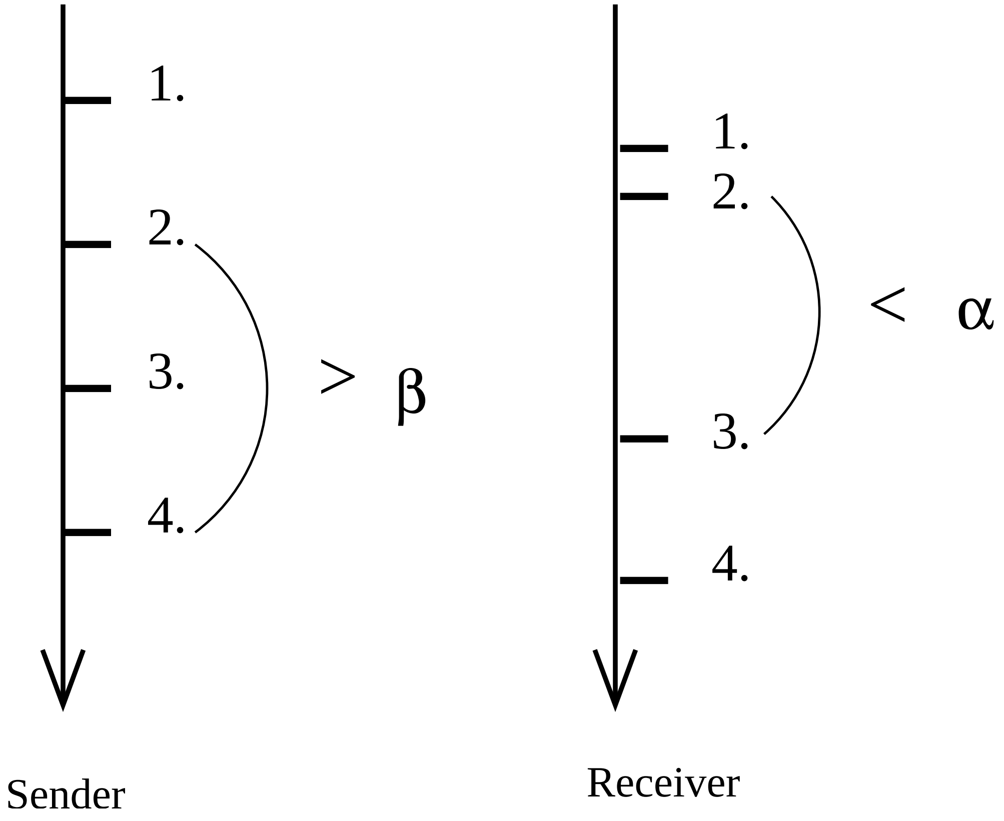

Figure 2: Timing requirements for the On-Line Deniable Authentication ZK protocol.

Timing Constraints:

1. P delays Step 4 until more than β time has elapsed on P's local clock since r_2 is received in Step 2.

2. V accepts the authentication received in Step 3 only if it is received within time α on V's local clock from when r_2 is sent in Step 2.

See Figure 2. The adversary is constrained by the (α, β) constraint. Here the constraint is needed for the security of *both* parties.

Theorem 4.2 *Protocol 10 is sound.*

Proof. Let P^* try to impersonate P. Suppose P^* sends $C(r_1)$ at time s in an interaction with a particular V. Either $C(r_1)$ appeared as a message sent in some previous execution by the real P or not. If not, then P^* will with all but negligible probability never get a signature from P on a string with prefix $C(r_1)$. So assume such a message was sent by the real prover P, necessarily at some time before s. (This allows us to concentrate on one particular execution and ignore all the rest.) Let V respond with r_2 at time $s' > s$. The only signature that V will accept at Step 4 is on $C(r_1) \circ r_2$, so P^* must get from P a signature on $C(r_1) \circ r_2$. This forces P^* to send r_2 to P at some time $s'' > s'$. But then P will not release r_1 before $s''+$ the time for β to elapse on P's local clock. Without knowing r_1, P^* cannot guess $auth_{r_1 \oplus r_2}(m')$ for any message m' other than the message

m that P authenticates with $r_1 \oplus r_2$. Thus, the only message that P^* can authenticate to V with $r_1 \oplus r_2$ in a timely fashion (by time $s'+$ the time for α to elapse on V's local clock) is the message m authenticated by P. $\perp$

Protocol 10 *cannot be* zero-knowledge because of the signature on $C(r_1) \circ r_2$. However, we prove the weakened form of deniability defined above:

Theorem 4.3 *For any polynomial time adversary controlling verifiers $V_1, V_2, \ldots V_k$ there exists a polynomial time simulator $\mathcal{S}$ such that if $\mathcal{S}$ is communicating with a prover willing to authenticate* some *sequence of messages $m_1, m_2, \ldots m_\ell$, then $\mathcal{S}$ can produce a transcript indistinguishable from an actual transcript.*

Proof. $\mathcal{S}$ will play a person-in-the-middle between the verifiers and the true prover. Following each Step 2 of a verifier V_i the simulator $\mathcal{S}$ freezes $\mathcal{A}$ until it receives r_1^i in Step 4 as well as the signature from the real prover. Once it has this information $\mathcal{S}$ can compute $auth_{r_1^i \oplus r_2^i}(\cdot)$ for any message and and V_i will accept the proof. $\perp$

Remark 4.4 *The use of timing can be partially replaced by employing moderately hard functions. In the case of Protocol 10 we could use moderately hard functions for commitment in Step 1 or for signing in Step 4. For the latter we should use the shortcut in the terminology of [29] and in this case we can actually make the protocol zero-knowledge. For soundness the verifier would require the Step 4 message to be received within a reasonably short time after the Step 2 message was sent: the delay should be considerably less than τ, the time to generate a "signature" without the short-cut information. Thus, the real time required for α to elapse on V's clock in any concurrent execution must be less than the real time required for β to elapse on P's clock in any execution; and the time for β to elapse in any concurrent execution on P's clock must be less than τ.*

Acknowledgment The authors are indebted to Russell Impagliazzo for motivating discussions, and to Oded Goldreich, Mihir Bellare, and Shafi Goldwasser for several valuable conversations. WE thank the anonymous referees for the usefull and enlightening comments

References

[1] M. Agrawal, N Kayal, and N. Saxena, *Primes in P.*

[2] E. Bach and J. Shallit, **Algorithmic Number Theory: Efficient Algorithms**, MIT Press, Cambridge, MA, 1996.

[3] B. Barak, *How to Go Beyond the Black-Box Simulation Barrier*, Proc. of the 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 106–115.

[4] M. Bellare and S. Goldwasser. *Encapsulated key escrow.* Manuscript, November 1996. Earlier version was MIT Laboratory for Computer Science Technical Report 688, April 1996.

[5] M. Bellare, M. Jakobsson and M. Yung. Round-optimal zero-knowledge arguments based on any one-way function. Advances in Cryptology- EUROCRYPT '97 Proceedings, Lecture Notes in Computer Science, Vol. 1233, Springer, 1997, pp. 280–305.

[6] M. Bellare and M. Yung. *Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation*, Journal of Cryptology, 9(3):149-166, 1996.

[7] M. Ben-Or, S. Goldwasser, and A. Wigderson, *Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation*, Proc. 20th ACM Symp. on Theory of Computing, 1988, pp. 1–10.

[8] T. Beth and E. Desmedt, Identification Tokens - or: Solving the Chess Grandmaster Problem. Advances in Cryptology–CRYPTO '90, Lecture Notes in Computer Science, Vol. 537, Springer, 1991, pp. 169–177.

[9] M. Blum. *Coin flipping by telephone: A protocol for solving impossible problems.* In Allen Gersho, editor, Advances in Cryptology: A Report on CRYPTO 81, pages 11-15, 24-26 August 1981. Department of Electrical and Computer Engineering, U. C. Santa Barbara, ECE Report 82-04, 1982.

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

[11] M. Blum, A. De Santis, S. Micali, and G. Persiano. *Noninteractive zero-knowledge*, SIAM Journal on Computing, 20(6), 1991, pp. 1084–1118.

[12] Blum M., P. Feldman and S. Micali, *Non-Interactive Zero-Knowledge Proof Systems*, Proc. 20th ACM Symposium on the Theory of Computing, Chicago, 1988, pp. 103–112.

[13] D. Boneh and M. Naor, *Timed commitments*, Advances in Cryptology - CRYPTO 2000, Lecture Notes in Computer Science, Vol. 1880, Springer, 2000, pp. 236–254.

[14] G. Brassard, D. Chaum, and C. Crépeau. *Minimum disclosure proofs of knowledge.* Journal of Computer and System Sciences, 37(2), 1988, pp. 156–189.

[15] G. Brassard, C. Crépeau and M. Yung, *Constant-Round Perfect Zero-Knowledge Computationally Convincing Protocols* . Theoretical Computer Science 84, 1991, pp. 23–52.

[16] S. Brands and D. Chaum, *Distance-Bounding Protocols* Advances in Cryptology – EUROCRYPT'93, 1993, Lecture Notes in Computer Science, Vol. 765, Springer Verlag, 1994, pp. 344–359.

[17] R. Canetti, *Universally Composable Security: A New Paradigm for Cryptographic Protocols*, Proc. of 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 136–145.

[18] R. Canetti, C. Dwork, M. Naor, R. Ostrovsky, *Deniable Encryption*, Advances in Cryptology – Crypto'97 Proceeding, Springer-Verlag, 1997, pp. 90–104.

[19] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali, *Resettable zero-knowledge*, Proc. of the Thirty-Second Annual ACM Symposium on Theory of Computing, Portland, Oregon, May 2000. Updated version available at the Cryptology ePrint Archive, record 1999/022, http://eprint.iacr.org/.

[20] R. Canetti, J. Kilian, E. Petrank, A. Rosen *Black-Box Concurrent Zero-Knowledge Requires* $\Omega(\log n)$ *Rounds*, SIAM Journal on Computing 32(1), 2002, pp. 1–47.

[21] D. Chaum and H. van Antwerpen, *Undeniable Signatures*, Advances in Cryptology – CRYPTO'89, Lecture Notes in Computer Science, Vol. 435, Springer-Verlag, 1990, pp. 212–216.

[22] D. Chaum, E. van Heijst and B. Pfitzmann, *Cryptographically strong undeniable signatures, unconditionally secure for the signer*, Advances in Cryptology - CRYPTO'91, Lecture Notes in Computer Science 576, Springer Verlag, 1992, pp. 470–484.

[23] R. Cramer and I. Damgård, *New Generation of Secure and Practical RSA-Based Signatures*, Advances in Cryptology - CRYPTO '96, Lecture Notes in Computer Science 1109, Springer Verlag, 1996, pp. 173-185.

[24] R. Cramer and V. Shoup *A Practical Public-Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack*, Advances in Cryptology - CRYPTO'98 Lecture Notes in Computer Science No. 1462, Springer Verlag, 1998, pp. 13–25.

[25] I. Damgård, Concurrent Zero-Knowledge is Easy in Practice, Theory of Cryptography Library, Record 99-14, July 1999. Available: `http://philby.ucsd.edu/1999.html`

[26] G. Di Crescenzo, Y. Ishai and R. Ostrovsky, *Non-Interactive and Non-Malleable Commitment*, Proc. 30th Annual ACM Symposium on the Theory of Computing, Dallas, 1998, pp. 141–150.

[27] G. Di Crescenzo and R. Ostrovsky, *On Concurrent Zero-Knowledge with Pre-processing*, Advances in Cryptology - CRYPTO'99 Lecture Notes in Computer Science, Vol. 1666, Springer, 1999, pp. 485–502.

[28] D. Dolev, C. Dwork and M. Naor, *Non-malleable Cryptography*, Preliminary version: Proc. 21st STOC, 1991. Full version: to appear, Siam J. on Computing. Available: http://www.wisdom.weizmann.ac.il/ naor

[29] C. Dwork and M. Naor, *Pricing via Processing -or- Combatting Junk Mail*, Advances in Cryptology – CRYPTO'92, Lecture Notes in Computer Science 740, Springer Verlag, 1993, pp. 139–147.

[30] C. Dwork and M. Naor, *Method for message authentication from non-malleable crypto systems*, US Patent No. 05539826, issued Aug. 29th 1996.

[31] C. Dwork and M. Naor, *An Efficient Existentially Unforgeable Signature Scheme and its Applications*, Journal of Cryptology, vol 11, 1998, pp. 187–208.

[32] C. Dwork and M. Naor, *Zaps and Their Applications* Proceedings 41st Annual Symposium on Foundations of Computer Science, 2000 pp. 283–293.

[33] C. Dwork, M. Naor, O. Reingold, and L. Stockmeyer, *Magic Functions*, J. ACM 50(6), 2003, pp. 852–921.

[34] C. Dwork, M. Naor, and A. Sahai, *Concurrent Zero-Knowledge*. Proceedings of the 30th ACM Symposium on the Theory of Computing, 1998, pp. 409–418.

[35] C. Dwork and A. Sahai, *Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints*, Lecture Notes in Computer Science, Vol. 1462, Springer Verlag, 1998, 442–457.

[36] C. Dwork, R. Shaltiel, A. Smith, and L. Trevisan, An Analysis of a Two-Round Zero-Knowledge Protocol, *Proc. 1st Theory of Cryptography Conference*, Springer Verlag, 2003, pp. 101–120.

[37] U. Feige, *Alternative Models for Zero Knowledge Interactive Proofs*, Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel, 1990.

[38] U. Feige, A. Fiat and A. Shamir, *Zero Knowledge Proofs of Identity*, J. of Cryptology 1 (2), 1988, pp. 77-94.

[39] U. Feige and A. Shamir, *Witness Indistinguishable and Witness Hiding Protocols* Proc. 22nd ACM Symposium on the Theory of Computing, 1990, pp. 416–426.

[40] U. Feige and A. Shamir, *Zero Knowledge Proofs of Knowledge in Two Rounds*, Advances in Cryptology – CRYPTO'89, Lecture Notes in Computer Science 435, Springer-Verlag, 1989, pp. 526–544.

[41] U. Feige, D. Lapidot and A. Shamir, *Multiple Non-Interactive Zero-Knowledge Proofs Based on a Single Random String*, Proceedings of 31st IEEE Symposium on Foundations of Computer Science, 1990, pp. 308-317.

[42] R. Genaro, H. Krawczyk and T. Rabin, *RSA-Based Undeniable Signatures*, Advances in Cryptology – CRYPTO'97, Lecture Notes in Computer Science 1294, Springer-Verlag, 1997, pp. 132–149.

[43] O. Goldreich, **Foundations of Cryptography, Volume 1** Cambridge University Press, 2001.

[44] O. Goldreich, *Concurrent Zero-Knowledge With Timing, Revisited,* Proceedings of the 34th ACM Symposium on Theory of Computing, 2002, pp. 332-340.

[45] O. Goldreich S. Goldwasser and S. Micali, *How to Construct Random Functions*, J. of the ACM 33 (1986), pp. 792–807.

[46] O. Goldreich, S. Goldwasser and S. Micali, Interleaved Zero-Knowledge in the Public-Key Model, Theory of Cryptography Library, Record 99-15, July 1999. Available: `http://philby.ucsd.edu/1999.html`

[47] O. Goldreich and H. Krawczyk. *On the Composition of Zero Knowledge Proof Systems.* SIAM Journal on Computing, Vol. 25, No. 1, pp. 169–192, 1996.

[48] O. Goldreich, A. Kahan, *How to Construct Constant-Round Zero-Knowledge Proof Systems for NP*, *J. Cryptology 9*(3), pp. 167–190, 1996.

[49] O. Goldreich, S. Micali and A. Wigderson, *Proofs that Yield Nothing But their Validity, and a Methodology of Cryptographic Protocol Design*, J. of the ACM 38, 1991, pp. 691–729.

[50] O. Goldreich and Y. Oren. *Definitions and properties of zero-knowledge proof systems.* Journal of Cryptology, 7(1):1-32, Winter 1994.

[51] S. Goldwasser, S. Micali and C. Rackoff, *The Knowledge Complexity of Interactive Proof-Systems*, Siam J. on Computing, 18(1) (1989), pp 186-208.

[52] O. Goldreich, M. Micali, A. Wigderson, *How to play any mental game*, Proc. 19th ACM Symp. on Theory of Computing, 1987, pp. 218–229.

[53] O. Goldreich and E. Petrank, *Quantifying Knowledge Complexity*, Proceedings of 31st IEEE Symposium on Foundations of Computer Science, 1991, pp. 59–68.

[54] S. Goldwasser and S. Micali. *Probabilistic Encryption*, Journal of Computer and System Sciences, Vol. 28, April 1984, pp. 270–299.

[55] S. Goldwasser, S. Micali, and C. Rackoff, *The Knowledge Complexity of Interactive Proof Systems,* SIAM Journal on Computing, Vol. 18, 1 (1989), pp. 186-208.

[56] S. Goldwasser, S. Micali and R. Rivest, *A Secure Digital Signature Scheme*, SIAM Journal on Computing, Vol. 17, 2 (1988), pp. 281-308.

[57] Johan Håstad, Russell Impagliazzo, Leonid Levin, and Michael Luby. *Construction of pseudorandom generator from any one-way function,* SIAM Journal on Computing, 28, 1999, pp. 1364–1396.

[58] J. Katz, *Efficient and Non-Malleable Proofs of Plaintext Knowledge and Applications,* Advances in Cryptology – EUROCRYPT‘2003, Lecture Notes in Computer Science, Vol. 2656, Springer Verlag, 2003, pp. 211–228.

[59] J. Kilian and E. Petrank, *An Efficient Non-Interactive Zero-Knowledge Proof System for NP with General Assumptions*, Journal of Cryptology, Vol. 11(1), 1998, 1–27.

[60] J. Kilian and E. Petrank, *Concurrent zero-knowledge in poly-logarithmic rounds,* Proceedings of the thirty third annual ACM Symposium on Theory of Computing, 2001, 560–569.

[61] J. Kilian, E. Petrank and C. Rackoff, *Lower Bounds for Zero Knowledge on the Internet*, Proceedings of 39th IEEE Symposium on Foundations of Computer Science, 1998, pp. 484–492.

[62] P. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems*, Advances in Cryptology - CRYPTO '96, Lecture Notes in Computer Science, Vol. 1109, Springer-Verlag, 1996, pp. 104–113.

[63] H. Krawczyk and T. Rabin, *Chameleon Hashing Signatures*, manuscript

[64] M. Naor, *Bit Commitment Using Pseudo-Randomness*, Journal of Cryptology, vol 4, 1991, pp. 151–158.

[65] M. Naor, *Deniable Ring Authentication*, Advances in Cryptology - CRYPTO '2002, Lecture Notes in Computer Science, Springer, 2002, pp. 481–498.

[66] M. Prabhakaran, A. Rosen, and A. Sahai, *Concurrent Zero Knowledge with Logarithmic Round Complexity*, Proc. 43rd IEE Symposium on Foundations of Computer Science, 2002, pp. 366–375.

[67] C. Rackoff and D. Simon, *Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack*, Advances in Cryptology - CRYPTO '91, Lecture Notes in Computer Science, Vol. 576, Springer-Verlag, 1992, pp. 433–444.

[68] R. L. Rivest, A. Shamir and D. A. Wagner, *Time- puzzles and time-release Crypto*, manuscript 1996. Available: http://theory.lcs.mit.edu/ rivest/RivestShamirWagner-timelock.ps

[69] R. Rivest, A. Shamir and L. Adleman, *A Method for Obtaining Digital Signature and Public Key Cryptosystems*, Comm. of ACM, 21 (1978), pp 120-126.

[70] R. Richardson and J. Kilian, *On the Concurrent Composition of Zero-Knowledge Proofs*, Advances in Cryptology - EUROCRYPT '99, Lecture Notes in Computer Science, Vol. 1592, Springer, 1999, pp. 415–431.

[71] A. Rosen, *A Note on the Round-Complexity of Concurrent Zero-Knowledge,* Advances in Cryptology - CRYPTO'2000, Lecture Notes in Computer Science, Vol. 1880, Springer, 2000, pp. 451–468.

[72] A. Sahai, *Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security.* Proc. of 40th ACM Symposium on Theory of Computing, 1999, pp. 543–553.