Concurrent Zero-Knowledge*

Cynthia Dwork* Moni Naor† Amit Sahai‡

October 4, 1999

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.* [6, 39]). 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 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 [22] 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 [38, 37], we consider two probability distributions in defining zero-knowledge:

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

*IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA. Research supported by BSF Grant 32-00032-1. E-mail: dwork@almaden.ibm.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: naor@wisdom.weizmann.ac.il.

‡MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA. 02139, USA. Most of this work performed while at the IBM Almaden Research Center. Also supported by a DOD NDSEG doctoral fellowship and partially by DARPA grant DABT63-96-C-0018. E-mail: amits@theory.lcs.mit.edu

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 oa 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* [38], 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. 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.

As is well known, graph Hamiltonicity can be proved in zero-knowledge using the sequential iteration of the following basic block, due to Manuel Blum [9]. 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.
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 permuation 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, there is no 3-round (black-box) zero-knowledge proof or argument for any language not in BPP that achieves negligible soundness error [35]. However, in order to parallelize the basic block it suffices for the verifier to commit to the vector of queries in advance:

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

For the simulation, we assume that the Verifier always produces valid Step 2 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. $V \longrightarrow P : K(\vec{q})$
2. $P \longrightarrow V : C(garbage)$
3. $V \longrightarrow P :$ open $\vec{q}$
2'. $P \longrightarrow V :$ Commit to graphs suitable for $\vec{q}$
3'=3. $V \longrightarrow P :$ open $\vec{q}$
4. $P \longrightarrow V :$ reply to queries

Thus, this protocol is zero-knowledge and yet can have arbitrarily low soundness error. Unfortunately, when Protocol Parallel-Hamiltonicity 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 Parallel-Hamiltonicity 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. The most deeply nested transaction, with V_n, is simulated roughly 2^n times. It has recently been shown that transcripts of this interleaving cannot be simulated in probabilistic polynomial time unless Hamiltonicity is in BPP [45]. And indeed, there exist protocols that are ordinarily zero-knowledge and yet fail dramatically to be zero-knowledge in the concurrent scenario [35].

Our Approach. 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: namely, the notion of time. We employ time in our protocols by imposing reasonable local timing contraints 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 $\alpha \leq \beta$): for any two (possibly the same) non-faulty parties 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 begins its measurement in real time after P_1 begins, then P_2 will finish after P_1 does.

Remark 1.1

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 6 and 7) discussed below.*

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 [23][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.

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 [40].

[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.

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 polymomial 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).

1.2 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 [22], 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 [22], "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 [22] 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 [15] and *chameleon signature* introduced by Krawczyk and Rabin [47], 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 [31] 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 [22], is analogous to that of *existential unforgeability under an adaptive chosen plaintext attack* for signature schemes [43], while taking care of "person-in-the-middle" attacks. The definition can be described informally as follows (see [22] 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 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 [22] (see also [24]). 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 [22] 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 1 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 1 was proved in [22] (Lemma 3.6) to be secure against *existential* forgery based on the non-malleability of E, Protocol 1 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 [28].) 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 2 Sequential ZK 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 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 2 remains zero-knowledge under concurrent executions, and in light of the results of [45] 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 $K_{g_1,g_2,p}(r)$ [16] 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.

[2]Almost all encryption schemes satisfy this property, however there are schemes designed for *deniable encryption* [14] that do not have it.

Protocol P: SeqZK 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.

1.3 Related work

The study of zero-knowledge in an aggregate was initiated in [22], 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 interaction may not actually yield a proof. The problem is the possible *malleability* of the interactive proof of knowledge, formalized and addressed in [22].

For a discussion of attempts to construct parallel (as opposed to concurrent) zero-knowledge protocols, see [3] and Goldreich, Chapter 6 [32]. Subsequent to the appearance of the conference version of our work [26], the problem of concurrent zero-knowledge was considered by several works, including those of Kilian, Petrank, and Rackoff [45] and Richardson and Kilian [52]. The first of these extends Goldreich and Krawczyk's results on the impossibility of a 3-round black-box zero-knowledge proof for membership in any language $L \notin BPP$, to show that there is no 4-round *concurrent* black-box zero-knowledge proof or argument for membership in any language $L \notin BPP$ [45]. The other shows the existence of (long) concurrent black-box zero-knowledge proofs for any language in NP [52].

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 [13], in which very accurate timing is needed in order to prevent person-in-the-middle attacks by distant processors (see also Beth and Desmedt [7]). Note that timing has been suggested as a cryptanalytic tool – the best example is Kocher's timing attack [46] – 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 [23], and their application to time-capsules was considered by Bellare and Goldwasser [1]. The notion of "time-lock puzzles" is discussed by Rivest, Shamir, and Wagner [50].

Summary of Results: We introduce timing in order to obtain zero-knowledge in concurrent executions.

We obtain four-round concurrent ε-knowledge interactive proofs and perfect concurrent zero-knowledge arguments for every language in NP under an (α, β) constraint. The protocol remains zero-knowledge independent of how many different theorems the prover (or provers) is proving in the concurrent interactions (Section 3). This construction can be modified to give concurrent perfect zero-knowledge arguments. In Section 4 we give several examples of protocols for deniable authentication. In particular, Protocols 6 and 7 are concrete and efficient concurrent solutions to the problem. When run without timing these yield sequential solutions *in the standard model.*

[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.

2 Model and Definitions

Timing. In our protocols, processors (or machines) use local clocks to measure elapsed time. In any execution an *adversary scheduler* has control over the timing of events, subject to an (α, β)-constraint (for some $\alpha \leq \beta$): for any two (possibly the same) non-faulty 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 if in addition P_2 begins its measurement in real time no sooner than P_1 begins, then P_2 will finish no sooner than P_1 does. The particular constraint may vary between protocols and must be stated explicitly as part of the description of any protocol.

Zero-Knowledge and Concurrent Zero-Knowledge We use the now standard "black box" forumulation of zero-knowledge proof systems and arguments [42, 37], an interactive proof system (P, V) for a language L is computational (or perfect) *zero-knowledge* if there exists a probabilistic, expected polynomial time oracle machine S, called the *simulator*, such that for every probabilistic polynomial time verifier strategy V^*, the distributions $(P, V^*)(x)$ and $S^{V^*}(x)$ are computationally indistinguishable (or identical) whenever $x \in L$. Here, formally, the machine V^* is assumed to take as input a partial conversation transcript, along with a random tape, and output the verifier's next response. This defintion also holds in the case of arguments or computationally-sound proofs, where the prover and verifier are both probabilistic polynomial time machines.

To investigate preservation of zero-knowledge in a distributed setting, we consider a probabilistic polynomial time adversary that controls many verifiers simultaneously. Here, the adversary $\mathcal{A}$ will take as input a partial conversation transcript of a prover interacting with several verifiers concurrently. Hence the transcript includes with each message sent or received by the prover, the local time on the prover's clock at which the event occurred. The output of $\mathcal{A}$ will either be a tuple $(\text{receive}, V, \alpha, t)$, indicating that P receives message α from V at time t on P's local clock, or (send, V, t), indicating that P must send a message to V at time t on P's local clock. The adversary must output a local time for P that is greater than all the times given in the transcript that was input to $\mathcal{A}$ (the adversary cannot rewind P), and standard well-formedness conditions must apply. If these conditions are not met, this corresponds to a non-real situation, so such transcripts are simply discarded. Note that we assume that if the adversary specifies a response time t for the prover that violates a timing constraint of the protocol with V, the prover should answer with a special null response which invalidates the remainder of the conversation with verifier V. The distribution of transcripts generated by an adversary $\mathcal{A}$ interacting with a prover P on common input x is denoted $(P \leftrightarrow \mathcal{A})(x)$.

An argument or proof system (P, V) for a language L is computational (or perfect) *concurrent zero-knowledge* if there exists a probabilistic, expected polynomial time oracle machine S such that for every probabilistic polynomial time adversary $\mathcal{A}$, the distributions $(P \leftrightarrow \mathcal{A})(x)$ and $S^{\mathcal{A}}(x)$ are computationally indistinguishable (or identical) whenever $x \in L$.

An argument or proof system (P, V) for a language L is computational *concurrent ε-knowledge* if for every $\varepsilon > 0$, there exists an oracle machine S, such that for every probabilistic polynomial time adversary $\mathcal{A}$, the running time of $S^{\mathcal{A}}$ is polynomial in n and $1/\varepsilon$, and any probabilistic polynomial time machine B that attempts to distinguish between the distributions $(P \leftrightarrow \mathcal{A})(x)$ and $S^{\mathcal{A}}(x)$ will have advantage at most ε whenever $x \in L$.

When the prover acts honestly and follows the protocol, it does not matter if there is a single entity that is acting as the prover for all verifiers, or if there are many entities that are acting as provers for subsets of the verifiers, since the actions of the provers would be the same, and in our model, the timing of events is controlled by the adversary.

String Commitment. A *string commitment* protocol between sender A and receiver B consists of two stages: (1) The *commit* stage: A has a string α to which she wishes to commit to B. She and B exchange messages. At the end of the stage B has some information that represents α, but B should gain no information about the value of α. (2)The *reveal* stage: at the end of this stage B knows α. The literature discusses two types of bit or string commitment: *strong-sender* and *strong-receiver*. In strong-sender string commitment there is only one possible way of opening the commitment. Such a scheme is designed to be secure against a probabilistic polynomial time receiver and an arbitrarily powerful sender. In strong-receiver commitment it is possible to open the commitment in two ways, but the assumed computational boundedness of the sender prevents him from finding a second way. Such a scheme is designed to be secure against an arbitrarily powerful receiver and a probabilistic polynomial time prover. See [22] for a formal definition of strong-sender commitment.

We require that α be semantically secure to B. Roughly speaking, this means that B has no "useful" information about α. Clearly, if the commitment is information-theoretic it is semantically secure. For the case of strong-sender security, we follow [22], and specify what this means using the notions of security with respect to relations (as shown in [22], this is equivalent to the traditional definition of semantic security [41]). The following discussion is from [22].

Let $\mathcal{A}$ be an adversary that produces a distribution $\mathcal{M}$ on strings of length $\ell(n)$ computable in probabilistic polynomial time. A string $\alpha \in_R \mathcal{M}$ is chosen and $\mathcal{A}$ receives $\text{hist}(\alpha)$, where hist is a probabilistic polynomial time computable function. The commitment protocol is executed where A follows the protocol and B is controlled by $\mathcal{A}$. The adversary $\mathcal{A}$ then produces a string β. We assume that the prefix of β is the description of $\mathcal{M}$.

$\mathcal{A}$ is considered to have succeeded with respect to R if $R(\alpha, \beta)$. Let $\pi(\mathcal{A}, R)$ be the probability that $\mathcal{A}$ succeeds with respect to R. The probability is over the coin-flips of $\mathcal{A}$, and the choice of α.

For the second probability, we have an adversary simulator $\mathcal{A}'$ who will not have access to the string commitment protocol, but has, however, the same computational power as $\mathcal{A}$. $\mathcal{A}'$ chooses a distribution $\mathcal{M}'$. An $\alpha \in_R \mathcal{M}'$ and $\text{hist}(m)$ is given to $\mathcal{A}'$. $\mathcal{A}'$ produces β. As above, $\mathcal{A}'$ is considered to have succeeded with respect to R if $R(\alpha, \beta)$ holds.

In information-theoretic string commitment, we require that the committer should with overwhelming probability be able to open the commitment in at most one way: any polynomial ability of the prover to find two openings of the commitment should translate into a polynomial probability of solving some underlying problem assumed to be intractable.

Non-Malleability. This was introduced in [22] as a natural strengthening of semantic security better suited to a distributed computing aggregate. Informally, in the context of encryption the additional requirement is that given the ciphertext it is impossible to generate a *different* ciphertext so that the respective plaintexts are related. The same concept makes sense in the contexts of string commitment and zero-knowledge proofs of possession of knowledge. In this paper we use non-malleable public-key cryptosystems resilient to the following chosen ciphertext attack in the post-processing mode [49]: The adversary has a ciphertext message $c \in E(m)$ she wishes to attack. She is given access to a decryption mechanism, to which she can make polynomially many adaptively chosen queries with one exception: she cannot submit the query c.

Malleability specifies what it means to "break" the cryptosystem. Informally, given a probabilistic polynomial time computable relation R (with certain trivial restrictions) and a ciphertext of a message α, the attacker is considered successful if she creates a *ciphertext* of β such that $R(\alpha, \beta) = 1$. The cryptosystem is non-malleable if for every attacker $\mathcal{A}$ there is an $\mathcal{A}'$ that, without access to the ciphertext of α, succeeds with similar probability as $\mathcal{A}$ in creating a ciphertext of γ such that $R(\alpha, \gamma) = 1$. A formal definition can be found in [22].

Deniable Authentication. We now summarize the requirements of a deniable authentication scheme, as discussed in Section 1.2. 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 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 simulatoted 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-knowldge 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-konwledge. 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 nore involved protocols that achieve this property but can can be based on general assumptions (existence of one-way functions)

3.1 Generic ε-Knowledge

We now show that adding timing constraints to Protocol Parallel-Hamiltonicity yields a protocol for concurrent ε-knowledge proofs of Hamiltonicity. Thus, every langauge in NP has a concurrent ε-knowledge proof. There is nothing magical about the Hamiltonicuty protocol; any "natural" prtocol for an NP-Compelte problem could also be used).

We assume the parameters for the information-theoretically secret commitment scheme K are part of the prover's public key. Commitments in Step 2 are computational (see, *e.g.*, [48]), 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 3 Generic Concurrent Zero-Knowledge Proof with Timing

1. $V \longrightarrow P$: *Commitment* $K(q)$ *to* m^2 *queries*
2. $P \longrightarrow V$: $C(Adj(\pi_1(G)), \ldots, Adj(\pi_n(G)))$
3. $V \longrightarrow P$: *Open queries* q
4. $P \longrightarrow V$: *Reply to all queries*

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. 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 3 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 4 and 5.

Theorem 3.1 *Protocol 3 is a computational concurrent ε-knowledge proof system for graph Hamiltonicity.*

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

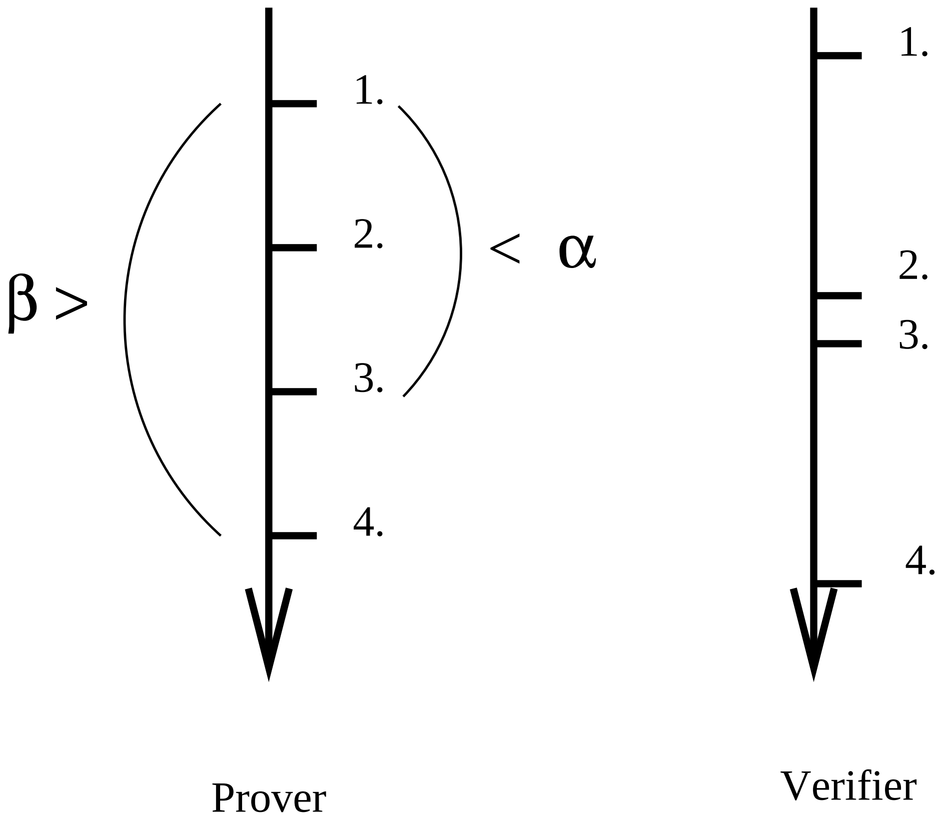

Figure 1: The timing requirements for the generic ZK protocol.

Concurrent ε-Knowledge. 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 $E_1, E_2, \ldots, E_{j-1}$, it starts E_j and tries to extract q_j. We call E_j the *current interaction.* At this point each of $E_1, \ldots, E_{j-1}$ 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 2) 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 3 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 1). 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 1 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 2 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 $\varepsilon/4t^3$.

We now discuss the implication of a waking conditionally delinquent verifier on the runnng time of the simulator and on the closeness of the simlated 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 3). 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 1 message)*. If $p_i \leq \varepsilon/4t^3$ then V_i is delinquent.

We 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 $o(1)$. Assuming that $\mathcal{A}$ cannot distinguish encryptions of random noise from encryptions of valid permutations of the graph, 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 2 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 3 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 3 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 3 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 negligibily 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. $\square$

Remark 3.2 *In the case of arguments, the use of timing in Protocol 3 can be partially eliminated by employing moderately hard functions. For example, the verifier can use a moderately hard function for commitment in Step 1 (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 2 message to be received within a reasonably short time after the Step 1 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 Perfect Zero-Knowledge Arguments Based on DLP

In this section we modify Protocol 3 to obtain Protocol 4 below, for which there exists a simulator that yields a distribution which is identical to the one produced in real interactions (no dependency on ε). The version described below assumes that the prover P is polynomial-time bounded, and is thus an argument-system (rather than a proof system). Protocol 4 requires five rounds of communication, and requires the Discrete Log Assumption.

We were "only" able to prove ε-knoweldge (and not zero-knowledge) for Protocol 3 because, in the simulation, P's Step 2 message had to be changed according to V's response in Step 3. We design Protocol 4 (as well as Protocol 5) to avoid this problem. In these protocols, the simulator only needs to change *future* messages based on the verifier's messages; for instance in Protocol 4, the simulator chooses P's Step 5 response based on the verifier's Step 4 messages. Of course, if the simulator were able to determine the Step 5 response based on a single Step 4 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 4 responses in order to be able to generate a Step 5 response that "fools" the verifier (and the Step 5 response will work for *any* valid Step 4 message of the verifier). As a result, the simulation can build a transcript step-by-step, never needing to go back to change some part of the transcript constructed up to that point. The simulator needs to rewind only to extract information from the verifier, not to alter P's communications.

In order to ensure that the extraction (which involves rewinding) for some verifier V does not get stuck because of some other verifier V', we add timing constraints. Under these constraints, while the simulator is waiting for the Step 4 message of V it will never have to produce a Step 5 message for some verifier V' whose Step 4 information has not already been extracted. The details follow.
In the sequel, the primes p_P and p_V are k bits long and hence are in the interval $[2^{k-1}, 2^k - 1]$. The strong-receiver commitment protocol K based on discrete logarithm used in the protocol has been described in the Introduction.

Protocol 4 Concurrent (Perfect) ZK Argument with Timing

1. $P \longrightarrow V$: *Choose random primes* p_P, q_P*, and* g_P *of order* q_P *in* $\mathbb{Z}^*_{p_P}$. *Choose random* $b \in \mathbb{Z}_{q_P}$. *Set* $K_P = (p_P, g_P, g_P^b \bmod p_P)$. *Send* $K_P, K_P(r_P)$ *for* $r_P \in_R \mathbb{Z}_{q_P}$.
2. $V \longrightarrow P$: *Choose random primes* p_V, q_V*, and* g_V *of order* q_V *in* $\mathbb{Z}^*_{p_V}$. *Choose random* g_V *of order* q_V *in* $\mathbb{Z}^*_{p_V}$. *Choose random* $a, a' \in \mathbb{Z}_{q_V}$. *Set* $h_V = g_V^a \bmod p_V$, $h'_V = g_V^{a'} \bmod p_V$. *Set* $K_V = (p_V, g_V, h_V)$. *Send* K_V, h'_V, $K_P(r_V)$ *where* $r_V \in_R \mathbb{Z}_{q_V}$.
3. $P \longrightarrow V$: K_V*(graphs); open* r_P
4. $V \longrightarrow P$: *Open* r_V*; Also, set* $r := r_V + r_P \bmod q_V$ *and send* $(ra + a') \bmod q_V$ *and a vector* $\vec{q}$ *of queries.*

5. *$P \longrightarrow V$: Check $(h_V)^r \cdot (h'_V) = (g_V)^c \bmod p$. Reply to queries and send b.*

Timing Constraints:

1. P requires the Step 4 message be received within α local time from receipt of the Step 2 message;
2. P delays the Step 5 message until at least β local time has elapsed since the receipt of the verifier's Step 4 message.

See Figure 2. For the proof to be zero-knowledge we require that the adversary be constrained by the (α, β)-constraint.

Theorem 3.3 *Protocol 4 is sound and (perfect) concurrent zero knowledge.*

Proof. As usual, completeness follows by inspection.

Ordinary Zero Knowledge. Before proving concurrent zero knowledge we outline the simulation technique for proving ordinary zero knowledge in the standard model, without timing. The (traditional) simulation works as follows.
Step 1. The simulator executes Step 1 of the protocol. There is no difference in the message composed by the simulator at this step and the message composed by the real prover. Note that the simulator knows the trapdoor b for the scheme K_P that it has chosen.
Step 2. The verifier replies with a scheme K_V, having trapdoor a (which may or may not be known to the verifier), and h'_V. Let p_2 be the conditional probability with which the verifier responds to the Step 1 message in the simulation. Clearly, p_2 is also exactly the probability with which the verifier responds in real executions. If the verifier fails to respond then the simulator outputs the simulated Step 1 message and halts.
Step 3. The simulator commits to a random string using K_V and opens r_P. Again, the simulator's message at this step is distributed exactly as the prover's Step 3 message. Indeed, the entire simulated transcript up to this point is distributed exactly as a real 3-step transcript is distributed. Let T denote the simulated transcript of the first 3 steps.
Step 4. The verifier opens r_V and sends $(r \cdot a + a') \bmod q_V$ with exactly the same conditional probability that it does in real executions (the conditioning is on T). A *valid* Step 4 message is a syntactically correct message that passes the test in Step 5. If the verifier sends no message or sends an invalid message, then output the transcript (including the invalid Step 4 message, if any) and halt.
Extraction: Rewind V^* to just after Step 2 and interleave the following two procedures until the first one halts.

1. Repeat until the verifier replies with a *different* valid Step 4 message:

   Open $K_P(r_P)$ to any $r'_P \in_R Z_{q_P} \setminus \{r_P\}$ using trapdoor b chosen in the simulation of Step 1. Note that even though r_P and r'_P are drawn from Z_{q_P}, rather than from Z_{q_V}, it is very likely that $r_P a + a' \not\equiv r'_P a + a' \bmod q_V$.

   Solve $(r'_P a + a') \bmod q_V$ and $(r_P a + a') \bmod q_V$ for $a \bmod q_V$.

2. Find a by brute force.

Step 5: open replies to Step 4 queries at will using trapdoor a for K_V.

This completes the description of the simulator. We now compute the running time. Let η be the probability conditioned on the transcript through Step 2, that the verifier will send a valid Step 4 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 4 message during extraction given $r'_P \not\equiv r_P \bmod q_V$. Let $r_0 = r_P \bmod q_V$.

1. Case $p_P \leq q_V$; This case cannot occur because $p_V < 2p_P$ and $q_V | (p_V - 1)$. Together with the fact that p_V and q_V are both prime (so $q_V \neq p_V - 1$), the second condition implies that $q_V < \frac{p_V}{2}$. Together with the first condition this yields $q_V < p_P$.

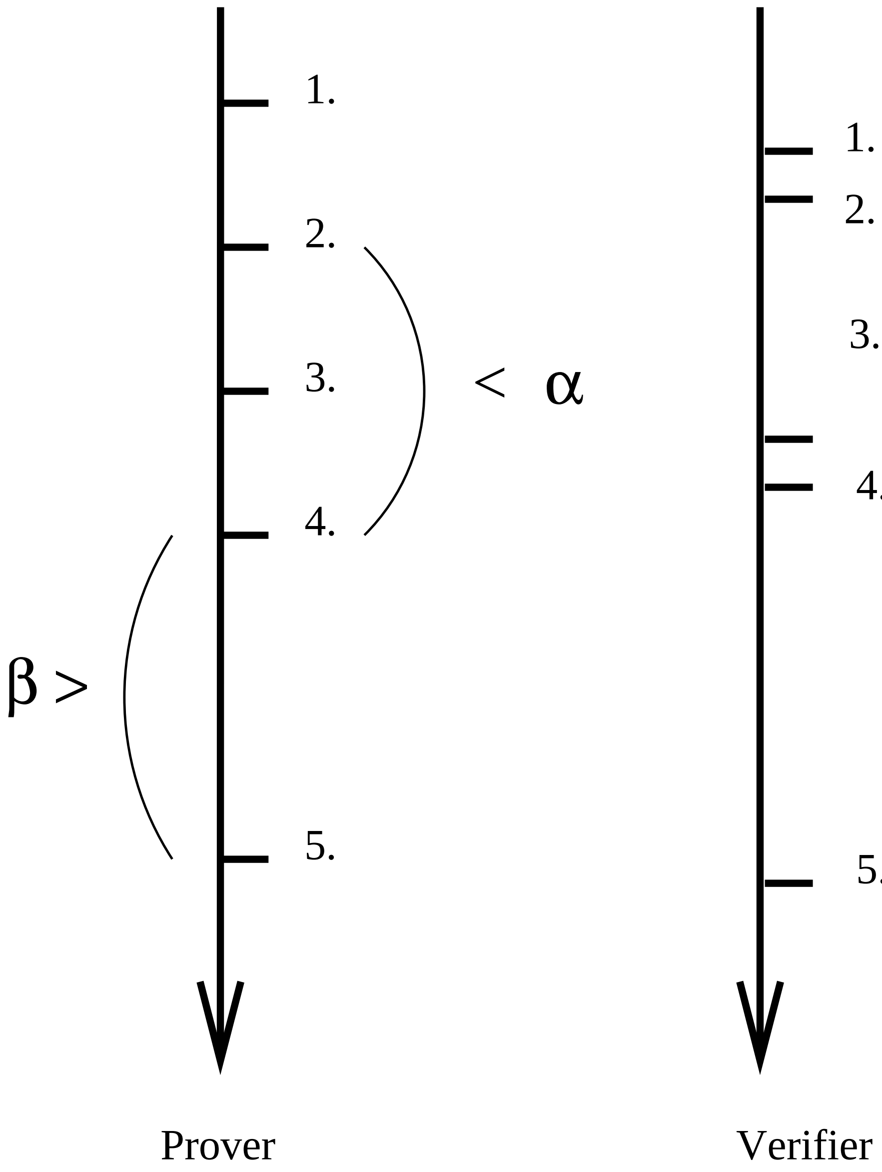

Figure 2: The timing requirements for the (Perfect) ZK protocol.

2. Since $p_P > q_V$, the probability of choosing an $r \equiv r_0 \bmod q_V$ is at most $\frac{2}{q_V}$. We consider two subcases:

- $\eta > \frac{3}{q_V}$: In this case $\eta - \frac{2}{q_V} > \frac{\eta}{3}$ and so $\xi \geq \frac{\eta}{3}$ and the expected number of trials during extraction is at most $\frac{3}{\eta}$. Thus the overall expected number of trials is at most $\eta\frac{3}{\eta} = 3$, leading to an expected $O(1)$ steps overall.
- $\eta \leq \frac{3}{q_V}$: In this case the brute force search succeeds in $O(q_V)$ steps, for a total of an expected $O(\eta q_V) = O(1)$ steps.

Concurrent Zero Knowledge The proof of concurrent zero knowledge is almost exactly the same as the proof for ordinary zero knowledge. We redefine a *valid* Step 4 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 transcript T through the Step 2 message of V_i, that the extraction procedure will be executed. This transcript T involves messages between the prover and other verifiers , and the probability is over all possible extensions of T. Finally, note that the timing constraints ensure that during the interaction with any verifier V_i the prover never needs to provide a Step 5 message to any V_j that sent its Step 4 message after V_i sent it Step 2 message. Thus the extraction will never get stuck because of another verifier[4].

The simulation is constructed message by message. If in some (P, V_i) interaction the adversary sends a valid Step 4 message for V_i, then the extraction procedure is executed for 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.

Soundness. There is no need to argue in the concurrent setting, since the parameters to K are used only once. The proof relies on the fact that, if P^* convinces V of a false statement with non-negligible probability, then by exploring the answers to two different query vectors (Steps 4 and 5), it is possible to obtain two different openings of a "committed" value.

Assume for the sake of contradiction that some probabilistic polynomial time bounded cheating P^* can convince V of a false theorem with non-negligible probability ρ. We argue that there exists a probabilistic polynomial time bounded M that, by interacting with P^*, can break the DLA with probability $\Omega(\rho^4)$, which is also non-negligible.

On input $p, q, g, h = g^a$, where g generates an order q subgroup of $\mathbb{Z}_p^*$, M interacts with P^* as follows.
1. P^* sends K_P, $K_P(r_P)$ to M.
2. M sets $p_V = p$, $q_V = q$, $g_V = g$. M chooses $c, r \in_R \mathbb{Z}_{q_V}$. M sends $K_V = (p, g_V, h)$, $h'_V = g_V^c h^{-r}$, and $K_P(r_V)$ to P^*, where $r_V \in_R \mathbb{Z}_{q_V}$.
3. P^* sends $K_V(graphs)$ and opens r_P. At this point M rewinds P^* to just after Step 1, sets $r_V = r - r_P \bmod q_V$, and M re-sends its original Step 2 message, modified so that the commitment $K_P(r_V)$ is to the new value of r_V.
2'. M sends $K_V = (p, g_V, h)$, $h' = g_V^c h^{-r}$, and $K_P(r_V$ to $P^*)$, where $r_V = r - r_P \bmod q_V$.

Assuming that P^* responds to the Step 2' message, one of two things can happen: either P^* changes the way it opens its Step 1 commitment to r_P or not. We examine the two cases separately.

Case P^* opens $r'_P \neq r_P$. In this case P^*'s message is:
3'. P^* sends $K_V(graphs')$ and opens $r'_P \neq r_P$. In this case M has extracted b (from 3' and 3).
4. Using the trapdoor b, M opens r_V to $r - r'_P \bmod q_V$. M also chooses a random query vector $\vec{q} \in_R \{0,1\}^n$ and gives to P^* the queries $\vec{q}$.
P^*'s check succeeds by definition of h'_V. P^* replies to the queries and sends b (which is already known). M then rewinds P^* to just before Step 4.
4'. Once again, using the trapdoor b, M opens r_V to $r - r'_P \bmod q_V$. M choose a random $\vec{q'} \in_R \{0,1\}^n$ and gives to P^* the queries $\vec{q'}$. Note that with overwhelming probability $q' \neq q$.

[4]It is an open question whether or not it suffices for P to ensure that time β has elapsed since the receipt of the Step 2 (rather than Step 4) message.

5. Once again, P^*'s check succeeds by definition of h'_V. P^* replies to the queries and sends b. M now has the reply to two different queries and hence the opening of some commitment under K_V (made in Step 3') to two different values. From this M can extract a. This completes the case P^* opens $r'_P \neq r_P$ in Step 3'.

Case P^* does not change r_P. Recall that in Step 2', M committed using K_V to $r_V = r - r_P \bmod q_V$.
3'. P^* sends $K_V(graphs')$ and opens r_P.
4. M opens r_V and sends a query vector $\vec{q} \in_R \{0,1\}^r$.
5. P^*'s check succeeds by definition of h'_V. P^* replies to the queries and sends b. M rewinds P^* to just before Step 4.
4'. M opens r_v to $r - r'_P \bmod q_V$ (it can do this because it has b). M then chooses a new $\vec{q'} \in_R \{0,1\}^n$ and sends $\vec{q'}$ to P^*.
5. P^*'s check succeeds by definition of h'_V. P^* replies to the queries and sends b. As in the previous case, M now has the reply to two different queries and hence the opening of some commitment under K_V (made in Step 3') to two different values. From this M can extract a.

We now analyze the probability of success of the extraction procedure. The (P^*, M) interaction can be viewed as a tree with probability ρ of success ("success" is when P^* answers all its questions correctly). After P^* sends its Step 1 message, with probability at least $1/2$ (over P^*'s choices in the the first message) the resulting node v_1 has probability at least $\rho/2$ of success. Assume we are in this "good" case. After M sends the Step 2 message, with probability at most $1-\rho/2$ there is immediate failure (P^* refuses to respond correctly), so with probability at least $\rho/2$, M will choose a Step 2 message to which P^* will respond correctly. Thus, the probability that M chooses two Step 2 message (once before and once after rewinding) to which P^* responds correctly is at least $\rho^2/4$, conditioned on being in a probability $\rho/2$ node after Step 1.

Assume P^* has answered correctly in both Step 3 and Step 3' and that the probability of success at v_1 is at least $\rho/2$. With probability at least $1/2$ the node v_3 reached after P^* sends its Step 3 message has probability at least $\rho/4$ of success. Arguing as above, the probability that M chooses two Step 4 message to which P^* responds correctly is at least $\rho^2/16$. Thus, the overall probability of extracting a is

$$\frac{1}{2}\frac{\rho^2}{4}\frac{1}{2}\frac{\rho^2}{16} = \Omega(\rho^4)$$

minus a negligible probability that $\vec{q} = \vec{q'}$ during the (P^*, M) interaction. $\square$

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. The protocol requires twelve rounds of communication. As written, this protocol requires that the prover be restricted to probabilistic polynomial time. We first recall the strong-sender commitment scheme of Naor [48], modified slightly 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 *([48]) 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 5 is based on several techniques:

- **Equivocal Commitment**. This is a property of the strong-sender commitment scheme of [48] first suggested in [20]: 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 [21].

- **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 5, 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 4: 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 5 Concurrent Computational ZK Argument - General Assumptions

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* $\vec{q}$

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 3.4 *Protocol 5 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 proof of Protocol 4 ....

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

$$g(s_1) \oplus g(s_2) \oplus g(s'_1) \oplus g(s'_2) = d'_1 \oplus d_1 \tag{1}$$

There are only 2^{4n} choices of the four seeds, and 2^{5n} choices for $d =_{\mathrm{def}} d_1 \oplus d'_1$, so with overwhelming probability over choice of d there do not exist seeds s_1, s_2, s'_1, s'_2 satisfying Equation 1.

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, [33]). 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. □

Remark 3.5 The following 6-round protocol, based on Feige and Shamir's idea of ORing the theorem to be proved with a (presumed) false statement [29], is an adaptation of the 4-round protocol of Bellare, Jakobsson, and Yung [3]. Although this protocol requires fewer rounds than Protocol 5, its communication and computation costs are higher.

We assume a "basic" proof for circuit satisfiability, that is actually a proof of knowledge of a satisfying assignment to the circuit. Let f be one-way function. We will OR the original circuit with a circuit ϕ_y that is satisfied when the prover knows x such that $y = f(x)$. If x is chosen by the verifier and not known to the prover, then this shouldn't help the prover to cheat. Note that commitments are used only in step 4, so the prover has plenty of time to send the pad for the commitment.

1. $V \longrightarrow P$: Choose pairs $(x_1^0, x_1^1), (x_2^0, x_2^1), \ldots (x_n^0, x_n^1)$; compute and send $(y_1^0, y_1^1), (y_2^0, y_2^1), \ldots (y_n^0, y_n^1)$ where $y_i^b = f(x_i^b)$.
2. $P \longrightarrow V$: Choose n random bits $b_1, b_2, \ldots b_n$ and send them.
3. $V \longrightarrow P$: Open $x_i^{b_i}$ for $1 \leq i \leq n$.
4. $P \longrightarrow V$: Commit to a circuit representing the original circuit ORed with $\bigwedge_{1 \leq i \leq n} \phi_i$, where ϕ_i is the circuit satisfiable iff there exists $x_i^{1-b_i}$ such that $y_i^{1-b_i} = f(x_i^{1-b_i})$.
5. $V \longrightarrow P$: Send challenges for the commitment in Step 4.
6. $P \longrightarrow V$: Respond to challenges.

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.

Proof Idea: Soundness follows from the proof of knowledge – extract either one of the unopened x_i's or the original proof. Extraction is possible because In Step 5 the verifier has complete freedom in choosing the challenges. Zero-knowledge follows from the fact that you can extract whp at least one x_i that wasn't opened in the main branch.

We strongly conjecture that this proof idea can be made rigorous.

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 Proof of Knowledge

In Protocol 6 below we modify Protocol 2 by adding timing constraints. These are essentially the same constraints that were used for the generic protocol (Protocol 3), 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 6 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 6 is sound and is concurrent* ε*-knowledge.*

Proof. The proof of soundness follows from the security of E, that it is non-malleable against a chosen ciphertext Post-processing attack, as was shown in Lemma 3.6 of [22].

The proof of concurrent ε-knowledge is essentially identical to the proof of of Theorem 3.1. □

Given the elegant and efficient non-malleable encryption scheme of Cramer and Shoup [18], based on the Decisional Diffie-Hellman assumption, Protocol 6 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 7 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 [43] for definitions and [17, 25] 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 7 On-Line Deniable AuthenticationUsing 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

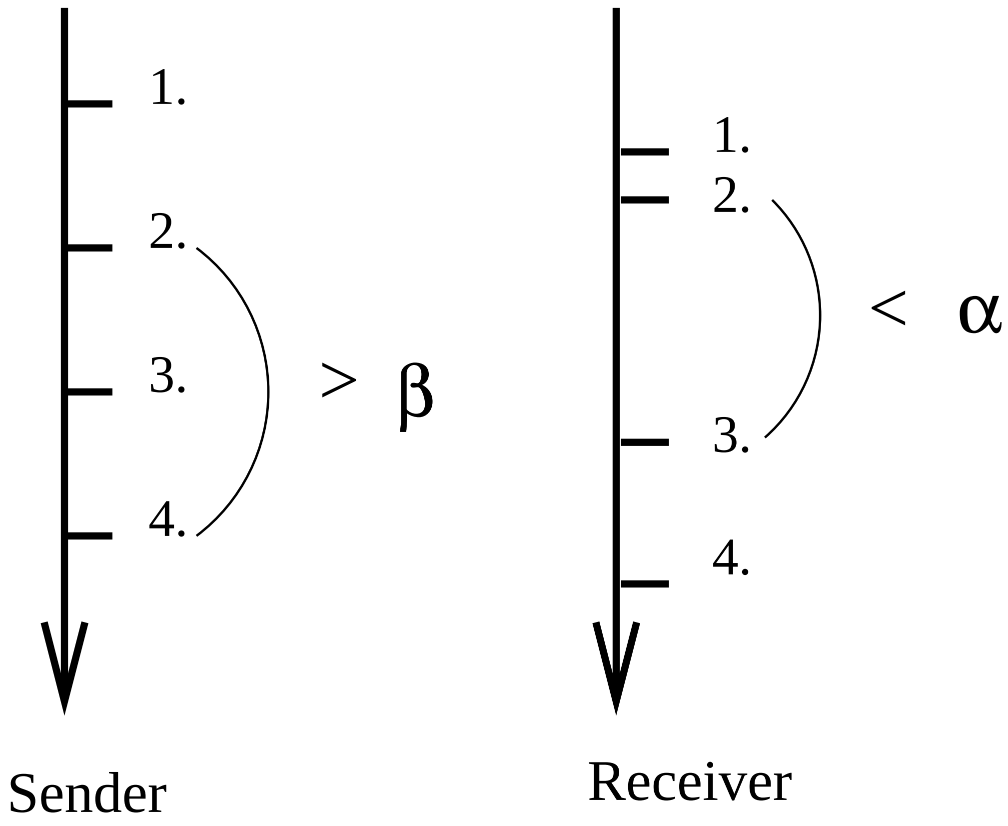

Figure 3: Timing requirements for the On-Line Deniable Authentication ZK protocol.

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

Theorem 4.2 *Protocol 7 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. $\square$

Protocol 7 *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. $\square$

Remark 4.4 *The use of timing can be partially replaced by employing moderately hard functions. In the case of Protocol 7 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 [23] 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.*

5 Recent Work

There have been several modifications to the original approach for achieving concurrent zero-knowledge [26]. We give a brief summary.

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 [27]. 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.

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 [34] proposed a model where the verifiers have public keys as well, but these public keys are simply registered. Damgård [19] 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.

Acknowledgment The authors are indebted to Russell Impagliazzo for motivating discussions, and to Oded Goldreich, Mihir Bellare, and Shafi Goldwasser for several valuable discussions.

References

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

[2] M. Bellare, R. Impagliazzo, and M. Jakobsson, *private communication*

[3] 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-Verlag, 1997, pp. 280–305.

[4] M. Bellare and P. Rogaway, *Provably Secure Session Key Distribution - The Three Party Case*, Proc. 27th STOC, 1995, pp 57–64.

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

[6] 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.

[7] 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-Verlag, 1991, pp. 169–177.

[8] 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.

[9] 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.

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

[11] 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.

[12] G. Brassard, C. Crepeau and M. Yung, *Constant-Round Perfect Zero-Knowledge Computationally Convincing Protocols* . Theoretical Computer Science 84, 1991.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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`

[20] 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.

[21] G. Di Crescenzo and R. Ostrovsky, *On Concurrent Zero-KNowledge with PRe-processing*, Advances in Cryptology - CRYPTO'99 Lecture Notes in Computer Science No. 1666, 1999, pp. 485–502.

[22] 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

[23] 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.

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

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

[26] 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.

[27] 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.

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

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

[30] 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.

[31] 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.

[32] O. Goldreich, **Foundations of Cryptography** (Fragments of a Book), 1995. Electronic publication: `http://www.eccc.uni-trier.de/eccc/info/ECCC-Books/eccc-books.html` (Electronic Colloquium on Computational Complexity).

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

[34] 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`

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

[36] 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.

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

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

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

[40] O. Goldreich and E. Petrank, *Quantifying Knowledge Complexity*, Proc. 31st FOCS, pp. 59–68, 1991

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

[42] 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.

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

[44] 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.

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

[46] 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.

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

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

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

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

[51] 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.

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