Concurrent Zero-Knowledge

Cynthia Dwork* Moni Naor† Amit Sahai‡

June 21, 1998

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*; for example, in the case of zero-knowledge interactive proofs or arguments, the interactions remain proofs but may fail to remain zero-knowledge. This paper addresses the problem of achieving concurrent zero-knowledge.

We introduce *timing* in order to obtain zero-knowledge in concurrent executions. We assume that the adversary is constrained in its control over processors' clocks by what we call an (α, β)-*constraint* for some $\alpha < \beta$: 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 obtain 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 efficient solutions.

1 Introduction

A distributed computing aggregate is a collection of physically separated processors that 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, 29]). In distinction with this traditional paradigm, processors in an aggregate do not in general 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 (but not always, *e.g.*, [4] in a different setting). In this paper we address this issue in the context of zero-knowledge interactions (and thus continue research initiated in [17] on zero-knowledge interactions in an aggregate.)

A zero-knowledge protocol is supposed to ensure that no information is leaked during its execution. However, when zero-knowledge interactions are executed *concurrently* both parties can be at risk. Consider the case of zero-knowledge proofs: 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. This is an issue of potential *malleability* of the interactive proof system, and is addressed in [17].

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

†Dept. of Applied Mathematics and Computer Science, 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. 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

The prover faces the risk that concurrent executions of a protocol by a single prover (with one or more verifiers) may leak information and may not be zero-knowledge *in toto*. (The problem is slightly more general than this; we elaborate in Section 2.) In this case the interaction remains a proof but may fail to remain zero-knowledge. To date, no zero-knowledge proof system has been proven zero-knowledge under concurrent execution. Indeed, recent work of Kilian and Petrank suggests that certain types of four-round interactive proof systems and arguments[1] cannot remain zero-knowledge under concurrent execution [35].

The situation is reminiscent of the case of *parallelizing* the iterations of a zero-knowledge interactive proof (P, V). Such a proof consists of a basic block that is iterated k times in order to ensure that the verifier will accept the proof of a false statement with a probability that shrinks exponentially in k. For example, if the basic block consists of

$P \longrightarrow V$: Commit to M_0, M_1
$V \longrightarrow P$: $b \in_R \{0,1\}$
$P \longrightarrow V$: open M_b

then the parallelization looks like

$P \longrightarrow V$: Commit to $M_0^1, M_1^1, \ldots, M_0^k, M_1^k$
$V \longrightarrow P$: $b_1 \ldots b_k \in_R \{0,1\}^k$
$P \longrightarrow V$: open $M_{b_1}, \ldots, M_{b_k}$

This natural parallelization is probably *not* zero-knowledge; indeed Goldreich and Krawczyk have shown that if L is any language having a three-round zero-knowledge (black-box simulatable) proof with negligible probability of error, then $L \in BPP$ (see [27] for this and analogous results for zero-knowledge arguments and constant round Arthur-Merlin games)..

In many cases the parallelization can be modified to achieve zero-knowledge by prepending a step in which the verifier commits to $b_1, \ldots, b_k$ (see [11, 23, 27]):

1. $V \longrightarrow P$: Commit to $b_1, \ldots, b_k$
2. $P \longrightarrow V$: Commit to $M_0^1, M_1^1, \ldots, M_0^k, M_1^k$
3. $V \longrightarrow P$: open $b_1, \ldots, b_k$
4. $P \longrightarrow V$: open $M_{b_1}, \ldots, M_{b_k}$

Since the verifier is completely committed to its queries before the prover sends any message, the simulation is easy: commit to random noise, wait until the verifier reveals its queries, then re-wind and choose a new sequence of commitments so that the — now known — queries $b_1, \ldots, b_k$ can be answered, and finish the simulation. To ensure that the prover's commitments at Step 2 are independent of the verifier's commitments at Step 1, the Step 1 commitments are information-theoretic.

In the concurrent scenario, 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. (There may be meta-methods, for example, involving certified public keys for the verifiers, but we do not consider such approaches here, and in any case at best such methods move the burden to some initial step.) Consider the following *nested* interleaving, shown in Diagram 1 below, of n colluding verifiers $V_1, \ldots, V_n$ following the generic four-round protocol described above with a single prover.

[1] In an *argument* the prover is polynomial time bounded.

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. It is a well-known open problem how to simulate transcripts with this interleaving in polynomial time; 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 is possible that the definition of zero-knowledge (*i.e.*, simulateable) is too demanding: interactions that leak no "useful" information may nonetheless not be simulateable. This is the philosophy in [23], and the motivation for *witness indistinguishability*. In the same spirit, we suggest three ideas for exploration:

1. introduce a notion of time to sufficiently restrict the behavior of an adversarial scheduler that the resulting execution is always simulateable (an *explicit* use of time);

2. use "moderately hard" functions to enforce timing constraints (an *implicit* use of time);

3. enhance the power of the simulator or relax the requirements on the output distribution.

Our precise notion of an explicit use of time is 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. An (α, β) constraint is implied by most reasonable assumptions on the behavior of clocks in a system (*e.g.* the linear drift assumption). The (α, β)-constraint is 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 III, IV and V). However, if the time-frame α is too short (not allowing some parties sufficient time to compute and send messages), then the completeness of those protocols is endangered.

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 [18][2]. 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.

[2] A short-cut behaves similarly to a trapdoor function, 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.

In the same spirit, 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 [30].

To understand 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).

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. (Without the timing constraint we are limited by an impossibility result of [35].) The protocol remains zero-knowledge independent of how many different theorems the prover (or provers) is proving in the concurrent interactions (Section 3). In Section 4 we give several examples of protocols for the case of *Deniable Authentication* (discussed next). In particular, Protocols IV and V are concrete and efficient solutions to the problem.

1.1 An Illustrative Example: Deniable Authentication

[17] 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 perhaps shouldn't!) be verifiable by a third party. To emphasize this last point we use the term *deniable authentication* and strengthen the definition in [17] by insisting on deniability.

Similar to a digital signature scheme, a deniable authentication scheme can convince V that $\mathcal{AP}$ is willing to authenticate m. However, unlike the case with 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 (other than what could be produced by V alone). 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 [13] and *chameleon signature* introduced by Krawczyk and Rabin [37], 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 [25] 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.

The notion of non-malleable security for a public key authentication scheme, defined in [17], is analogous to that of *existential unforgeability under an adaptive chosen plaintext attack* for signature schemes [33], but we must make sure to take care of *PIM* ("person-in-the-middle") attacks.

In terms of the discussion above, the definition of non-malleable security in this context is concerned with protecting the *verifier* against an imposter trying to impersonate the *prover* (authenticator) and falsely "authenticating" the message m. A deniable authentication protocol satisfying the 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 [17] (see also [19]). P's public key is E, chosen according to a non-malleable public key cryptosystem generator. (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 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 0: 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 proved in [17] to be non-malleably secure, Protocol 0 is not zero-knowledge. However, the protocol is easily modified to be zero-knowledge by the addition of a proof of knowledge. (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 [22].) Clearly, once it is zero-knowledge the interaction yields no "paper trail" of involvement by the prover/authenticator, under sequential executions by the same prover/authenticator.

For the modification we use an information-theoretic commitment scheme $K_{g_1,g_2,p}(r)$ [14] to commit to a string r. $K_{g_1,g_2,p}(x)$ is defined as follows: g_1, g_2 generate the same q-sized subgroup of $\mathbb{Z}_p^*$, where q is a large prime dividing $p-1$. Given g_1, g_2, and p, commit to x by sending $g_1^x g_2^z \bmod p$ where $z \in_R \mathbb{Z}_q$. Note that this is distributed uniformly in the subgroup generated by g_1 for all x. Decommitment is by revealing x and z. Note that if the committer knows a such that $g_2 \equiv g_1^a \bmod p$, then the "commitment" can be opened arbitrarily, so the security of the commitment relies on the hardness of finding discrete logarithms modulo p.

Protocol I: SeqZK Deniable Authentication

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

Intuitively, this protocol "should be" zero-knowledge because Step 2 yields no information about r *even information-theoretically*. However, standard simulation techniques fail for concurrent executions; the difficult transcripts are those in which the adversary nests many executions. We do not know if Protocol I remains zero-knowledge under concurrent executions by the same $\mathcal{AP}$. Similarly, if there is a collection of $\mathcal{AP}$s, all using the same secret key, concurrent executions of the protocol may not be zero-knowledge.

Related work. For a discussion of attempts to construct parallel zero-knowledge protocols, see [3] and Goldreich, Chapter 6 [26]. The problem of concurrent zero-knowledge was considered by several groups including Kilian and Petrank [35] and Bellare, Impagliazzo and Jakobsson [2]. The use of timing considerations to ensure soundness and zero-knowledge is new. The only work we aware of that uses timing in zero-knowledge protocols is Brands and Chaum [12], 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 [36] – 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 [18]. The application as time-capsule was considered by Bellare and Goldwasser [1]. The notion of "time-lock puzzles" is discussed by Rivest, Shamir, and Wagner [40].

2 Model and Definitions

Timing. In our protocols, processors (or machines) use local clocks to measure elapsed time. In any execution the adversary 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 In the original "black box" forumulation of zero-knowledge proof systems [32], 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 (receive, V, α, 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.

NIZK. In a non-interactive zero-knowledge (NIZK) proof [5, 9, 10, 24] the prover P and verifier V have a common input x and also share a random string σ, called the reference string, of length polynomial in the length of x. To convince the verifier of the membership of x in some fixed NP language L, the prover is allowed to send the verifier a single message $m = P(x, \sigma)$, computed (probabilistically) as a function of x and σ. The probabilistic polynomial time verifier must decide whether or not to accept as a function of x, σ, and m. Such an interaction (P, V) is an NIZK proof system for L if: (1) If $x \in L$, for all σ, $(P, V)(x, \sigma)$ accepts. (2) If $x \notin L$, for all P^*, the probability over σ and the random coins of P and V that $(P^*, V)(x, \sigma)$ accepts is negligible. (3) There exists a probabilistic polynomial time simulator S such that, if $x \in L$, then the distributions $S(x)$ and $(\sigma, P(x, \sigma))$, where in the latter distribution σ is chosen uniformly, are computationally indistinguishable. We further ask that the prover be probabilistic polynomial time, but also allow that in the case when $x \in L$, the prover is given a witness w for the membership of $x \in L$. The distribution $(\sigma, P(x, \sigma, w))$ must be computationally indistinguishable from $S(x)$ no matter how the witness w is chosen. We use the variant proposed by [24], in which any polynomial number of NP statements can be proven in zero-knowledge using the same reference string σ (see [34] for a more efficient implementation).

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: *computational* and *information-theoretic*. These terms describe the type of secrecy of the committed values offered by the scheme. In computational 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 information theoretic 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 [17] for a formal definition of computational commitment.

Non-Malleability. This was introduced in [17] 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 "Rackoff-Simon" attacks [39]: The adversary has a ciphertext message $\sigma \in E(m)$ 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 σ.

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 [17].

3 Proofs and Arguments

We now describe and prove correct a general protocol for concurrent ε-knowledge proofs of NP statements. The protocol can be modified to yield concurrent perfect zero-knowledge arguments. Without the timing constraints the protocol is the standard parallelization of the zero-knowledge proof for Hamiltonian Circuit [28] (any other zero-knowledge proof for an NP-complete problem could also be used). In that protocol, the prover chooses a random permutation π of the graph G and sends to the verifier for every pair of vertices i and j, a commitment to the existence or nonexistence of the edge (i, j). The verifier chooses a random bit q. If $q = 0$, the prover opens all the commitments and sends the permutation π; if $q = 1$, the prover reveals only those edges in a Hamiltonian circuit. The verifier either checks that the graph revealed really maps to G under π^{-1}, or that all the edges revealed really do form a Hamiltonian circuit.

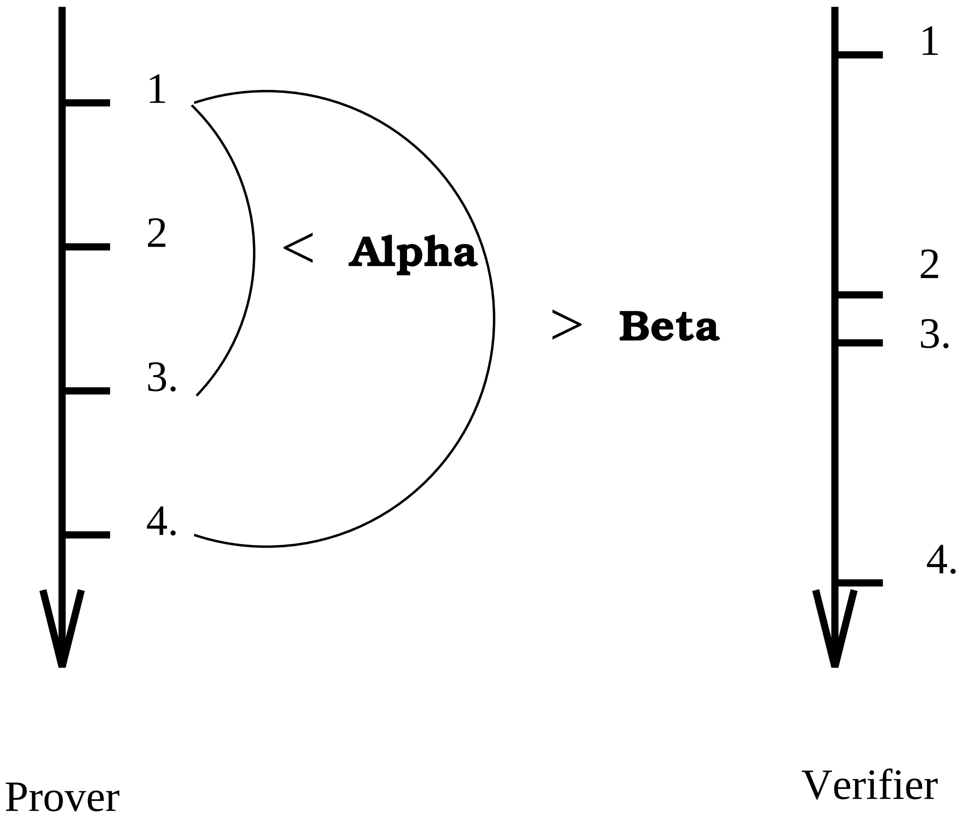

Figure 1: The timing requirements for the generic ZK protocol.

This is iterated n times, where n is the size of the graph or a security parameter.

In the parallelized version, all n executions will be run concurrently, with the verifier committing to the queries in advance in order to ensure zero-knowledge [27]. The commitments in Step 1 are information-theoretic, so even an arbitrarily powerful prover learns nothing about the committed values before the commitments are opened; moreover this implies that the Step 2 commitments are *a fortiori* independent of the Step 1 commitments. We assume the parameters for K are part of the prover's public key. Commitments in Step 2 are computational (see, *e.g.*, [38]), 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 II: Generic CoZKP with Timing

1. $V \longrightarrow P$: Commitment $K(q)$ to m^2 queries
2. $P \longrightarrow V$: C(graphs)
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 II 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 Protocol II'.

Theorem 3.1 *Protocol II is computational concurrent ε-knowledge and sound.*

Proof. The proof of soundness appears in the full paper. 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 hence in n), 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^2/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 ε/t^2. 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^2/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 $\varepsilon/2t$.

During the current interaction E_j the extraction is done as it was done for V_1: 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 this trial is not counted among the t^2/ε attempts to extract q_j. 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). As we shall see this does not occur with probability negligibly more than $\varepsilon/2t$.

We now discuss the implication of a waking conditionally delinquent verifier has on the run-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 $p_i \leq \varepsilon/2t$ then V_i is delinquent.

We compute the probability that the simulation will declare V_i conditionally delinquent if $p_i \geq \frac{\varepsilon}{2t}$. 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 $p_i \geq \frac{\varepsilon}{2t}$ then the probability that none of the t^2/ε attempts to extract q_i will succeed is at most $(1-\frac{\varepsilon}{2t})^{t^2/\varepsilon}$. Setting $x = t^2/\varepsilon$ yields $f(x) = t/2$, so the condition on $f(x)$ is satisfied when $\varepsilon = o(1)$, and we get a probability of $e^{-t/2}$. Thus, the probability of classifying V_i as conditionally delinquent when V_i is in fact *not* delinquent, is at most $e^{-t/2} + \mu$, 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 is that in the former no delinquent verifier ever wakes up. However, this is a low probability event: its probability is bounded by $t \cdot \frac{\varepsilon}{2t} = \varepsilon/2$.

The effect of the waking of conditionally delinquent verifiers on the run-time of the extraction procedure for V_j is relatively minor: a conditionally delinquent V_i will be the first to wake up in an expected $\varepsilon/2t$ fraction of the cases, adding an expected t aborted trials to the t^2/ε trials of V_j. $\square$

Protocol II can be modified to yield Protocol II' below, for which there exists a simulator that yields a distribution which is indistiguishable from 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). The distribution output by the simulator in this case is actually *identical* to the actual distribution on transcripts. Protocol II' requires 7 rounds of communication. A more subtle protocol, requiring only 5 rounds, appears in the full paper. Both protocols require the Discrete Log Assumption. The information-theoretic commitment protocol K used in the protocol has been described in the Introduction.

Protocol II': Generic Perfect CoZKA with Timing

0. $P \longrightarrow V$: primes p_P,q_P, and $g_P, h_P \in \mathbb{Z}^*_{p_P}$, where $ord(g_P) = q_P$, and $h_P = g_P^b \bmod p_P$, for $b \in_R \mathbb{Z}_{q_P}$.
1. $V \longrightarrow P$: $K_{g_P,h_P,p}(\vec{q})$ to all n queries, primes p_V, q_V, and $g_V, h_V, h'_V \in \mathbb{Z}_{p_V}$, where $ord(g_V) = q_V$, $h_V = (g_V)^a \bmod p_V$, and $h'_V = (g_V)^{a'} \bmod p_V$, for $a, a' \in_R \mathbb{Z}_{q_V}$
2. $P \longrightarrow V$: commitment using K_{g_V,h_V,p_V} to all n graphs, and $K_{g_V,h_V,p_V}(r_P)$, for $r_P \in_R \mathbb{Z}_{q_V}$
3. $V \longrightarrow P$: $r_V \in_R \mathbb{Z}_{q_V}$
4. $P \longrightarrow V$: Open r_P. Set $r := r_V + r_P \bmod q_V$
5. $V \longrightarrow P$: Reveal $\vec{q}$ and $c = ra + a' \bmod q_V$
6. $P \longrightarrow V$: Check $(h_V)^r \cdot (h'_V) = (g_V)^c \bmod p_V$. Reply to queries, and reveal b.

Timing Constraints: (1) P requires the Step 5 message be received within α (local) time from receipt of the Step 1 message; (2) P delays Step 6 until the entire interaction, from the receipt of the Step 1 message to the sending of the Step 6 message, takes at least β local time; for the proof to be zero-knowledge we require that the adversary be constrained by the (α, β)-constraint. For completeness we must assume that V can send the required messages in local time α.

In Step 2 the prover commits to the graphs and then initiates a random string selection subprotocol for choosing the string r used in Step 5 (explained below). Steps 3 and 4 are part of the string selection protocol. We prove the following theorem.

Theorem 3.2 *Protocol II' is sound and is concurrent perfect zero-knowledge.*

Proof. We give only a sketch of the proof of soundness; a complete proof appears in the full version. The proof of perfect concurrent zero knowledge is given in full.

Soundness. First, observe that if a commitment using $K_{g,h,p}$ is revealed in two different ways, say $g^{r_1}h^{s_1} = g^{r_2}h^{s_2}$, then the discrete log of h with respect to g can easily be recovered; in this case it would be $(r_1 - r_2)/(s_2 - s_1)$.

Formally, to argue soundness, we show that if any probabilistic polynomial time P^* can convince V of a false statement with probability ρ, then there exists a probabilistic polynomial time machine M that can break the DLA with probability $\rho^2 - \mu(n)$, for some negligible function μ. Here, we give an informal argument to this effect. Note that there is no need to argue this in a concurrent setting, since all parameters to the commitment scheme K are used only once for each (P, V) interaction. On input primes p, q, generator g of a q-order subgroup of $\mathbb{Z}_p^*$, and $h = g^a$, $M(p, q, g, h)$ proceeds as follows: M chooses $c, r \in_R \mathbb{Z}_q$, and lets $h' = g^c \cdot h^{-r}$. M uses P^* as a subroutine, interacting with P^* through Steps 0 and 1, using $p_V = p$, $q_V = q$, $h_V = h$, and $h'_V = h'$. M proceeds through Steps 2-4 with P^*, and receives r_P in P^*'s Step 4 message. M then rewinds P^* to Step 3, and sends $r_V = r - r_P$. If P^* then in Step 4 opens its commitment to something other than r_P, then M can immediately extract a and terminate. If P^*'s Step 4 response is the same as before, then M proceeds to Step 5, revealing its random queries and sending the value of c that it had chosen in the beginning. Note that P^*'s Step 6 check will pass by construction of h', and P^* will reply correctly to all queries, and supply b such that $g_P^b = h_P$. By assumption, we make it this far with probability roughly ρ. Now, M rewinds P^* to Step 5, and using knowledge of b, decommits to a new randomly chosen query vector, and gives the same c as before. Again by assumption, P^* again gives a correct Step 6 response with probability roughly ρ. Now, with all but negligible probability, there is some query bit position which is different in the second query vector than it was in the first. In order for P^* to have answered correctly both times on this bit, it must have revealed some edge differently in response to the first query than in the second. From this, M can recover a, and terminate. Thus, we have shown that with probability roughy ρ^2, M can break the DLA.

The proof of soundness above essentially relies on the fact that Steps 1-5 of Protocol II' are zero-knowledge for the Verifier with respect to a. The simulator is essentially contained in the machine M described above.

Concurrent Perfect Zero-Knowledge. We will show formally that Steps 1-5 are also a concurrent proof of knowledge of a. Essentially, this is the case because obtaining Step 5 responses from V for two different values of r, say r_1 and r_2, yields a linear system which can be solved for a and a'. For proving concurrent perfect zero-knowledge, we show how an expected polynomial-time simulator can in fact extract the discrete log a in this manner for every verifier, and use this to decommit in Step 6 to any desired value. The formal proof follows:

The timing constraints give us the following:

Interleaving Constraint: While any verifier is in Steps 1-5, no new interaction can be started and proceed to Step 6.

We define a *valid* Step 5 response to be one which is received within time α of Step 1 and for which P's Step 6 check succeeds, *i.e.*, $h_V \cdot (h'_V)^r = g_V^c \bmod p_V$. We now describe a subroutine of the simulator called `Extract`. The subroutine takes two arguments, the name of a verifier V_i, and a partial starting transcript T that includes the Step 1 message of V_i. `Extract(`V_i`,`T`)` is only called if the simulator already has obtained one valid Step 5 response from V_i. The purpose of calling `Extract` on V_i is to extract the $a = \log_{g_V}(h_V)$ fixed by V_i in Step 1. In `Extract(`V_i`,`T`)` the simulator repeats the following procedure *as many times as needed* to obtain another valid Step 5 response from V:

Starting with partial transcript T, run a simulation until either V_i gives a valid Step 5 response or more than time α has passed on P's local clock since V_i's Step 1 message was received. During this simulation:

(1) For any verifiers introduced after the Step 1 message of V_i, by the Interleaving Constraint, we know the simulator will never have to supply a Step 6 response in the time allotted, so simulate the interaction with them perfectly.

(2) If any verifier V_j, introduced before the Step 1 message of V_i gives a valid Step 5 response, then

- **(2a)** If the simulator has already extracted V_j's a, then the simulator can decommit to any response needed in Step 6, so it does so.
- **(2b)** If not, the simulator executes `Extract(`V_j`,`T`)`.

After executing `Extract(`V_i, T`)` we will have received two valid Step 5 responses from V_i. If the two responses received are to the same query r, the simulator breaks the discrete logarithm a by brute force, by trying all q possible values for a. It is clear that this happens with probability only $1/q$ in each trial: r_P is chosen randomly by the simulator, and since only an information-theoretically secret commitment to r_P is given to V, r_V can have no dependence on r_P. Hence $r_P + r_V \bmod q_V$ is truly random. If the simulator does receive valid responses to $r_1 \neq r_2$, the simulator uses the two responses $c_1 = r_1 a + a'$ and $c_2 = r_2 a + a'$ to solve for a. This can always be done since $\mathbb{Z}_q$ is a field. We call such a verifier, whose a is known, *neutralized.*

Now, to generate its transcript the simulator begins a simulation with the adversary. Whenever a verifier V that has not already been neutralized gives a valid Step 5 response, the simulator calls `Extract(`V, T`)`, where T is the partial transcript up to and including the Step 1 message of V. When `Extract` terminates, V has been neutralized and thus the simulator decommits in Step 6 to uniformly chosen valid responses to all queries. This continues until the simulation is concluded.

The distribution of transcripts produced by such a simulation is identical to those arising from an actual interaction with the adversary. It remains to be shown that the expected running time of the simulator is polynomially bounded. Each trial within *Extract*, not counting time taken by recursive function calls, certainly takes polynomial time, say $O(n^c)$. Let us analyze the expected running time of a function call to *Extract*. Now, conditioned on some partial transcript T, let X_i denote the random variable for the time to complete *Extract(*V_i, T*)*. Let p_i denote the probability over simulations starting at T that V_i will give its Step 5 response during a simulation trial. If V_1 is the first verifier appearing in T, then during *Extract(*V_1, T*)*, no recursive calls can be made for other verifiers. Hence, $X_1 \leq O(n^c)((1/q)q + 1 + p_1 + (1-p_1)(1+X_1))$, and so by linearity of expectation, $E(X_1) \leq O(n^c)\frac{3}{p_1}$. More generally, if V_i is the i'th verifier appearing in the transcript T, then $X_i \leq O(n^c)(2 + \sum_{j=1}^{i-1} p_j X_j + p_i + (1-p_i)(1+X_i))$, and a simple induction shows that $E(X_i) \leq O(n^c)\frac{3i}{p_i}$. Now, in the simulation, conditioned on a partial transcript T, the probability that *Extract(*V_i, T*)* will be called (from outside *Extract*) is exactly p_i. Thus, the expected amount of time the simulation will spend on *Extract(*V_i, T*)* is $O(n^c) \cdot 3i$. Since this does not depend on T, we can remove the conditioning and conclude that the expected amount of time the simulation will spend in `Extract` for V_i will be $O(n^c) \cdot 3i$. We note that the total number of verifiers that can be present in the final transcript is bounded by the amount of time the adversary runs for. Hence, if the adversary's running time is $t(n)$, the expected amount of time the simulation will spend in *Extract* for all of the verifiers is $O(n^c t(n)^2)$. The rest of the simulation will certainly take no more than expected $O(n^c)t(n)$ time, and so we conclude that the expected running time is polynomial in n.[3] □

Remark 3.3 *In the case of arguments, the use of timing in Protocol II can be partially eliminated by employing moderately hard functions.*

4 Protocols for Concurrent Deniable Authentication

Although the problem of achieving concurrent deniable authentication is solved by the general technique of the previous section, a further exploration of the problem yields several "special purpose" solutions interesting because they suggest very different techniques for coping with the problem of achieving zero-knowledge under concurrent executions that can be applied to other problems. Throughout this section we assume the prover P is polynomial time bounded (it is not clear what authentication means if this is not the case).

Timed Proof of Knowledge In Protocol III below we modify Protocol I by adding timing constraints. These are essentially the same as for the generic proof, 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. Recall that 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 [22]. The random strings $q_1, \ldots, q_{2n/3}$ are used to prove knowledge: intuitively, r can easily be computed from its inner product with n random strings, so the ability to compute the inner product of r with two thirds this many random strings still hides much of r but convinces the

[3] Note that if the adversary is expected polynomial time, but its running time has polynomially bounded variance, then the above analysis shows the simulation will also be expected polynomial time.

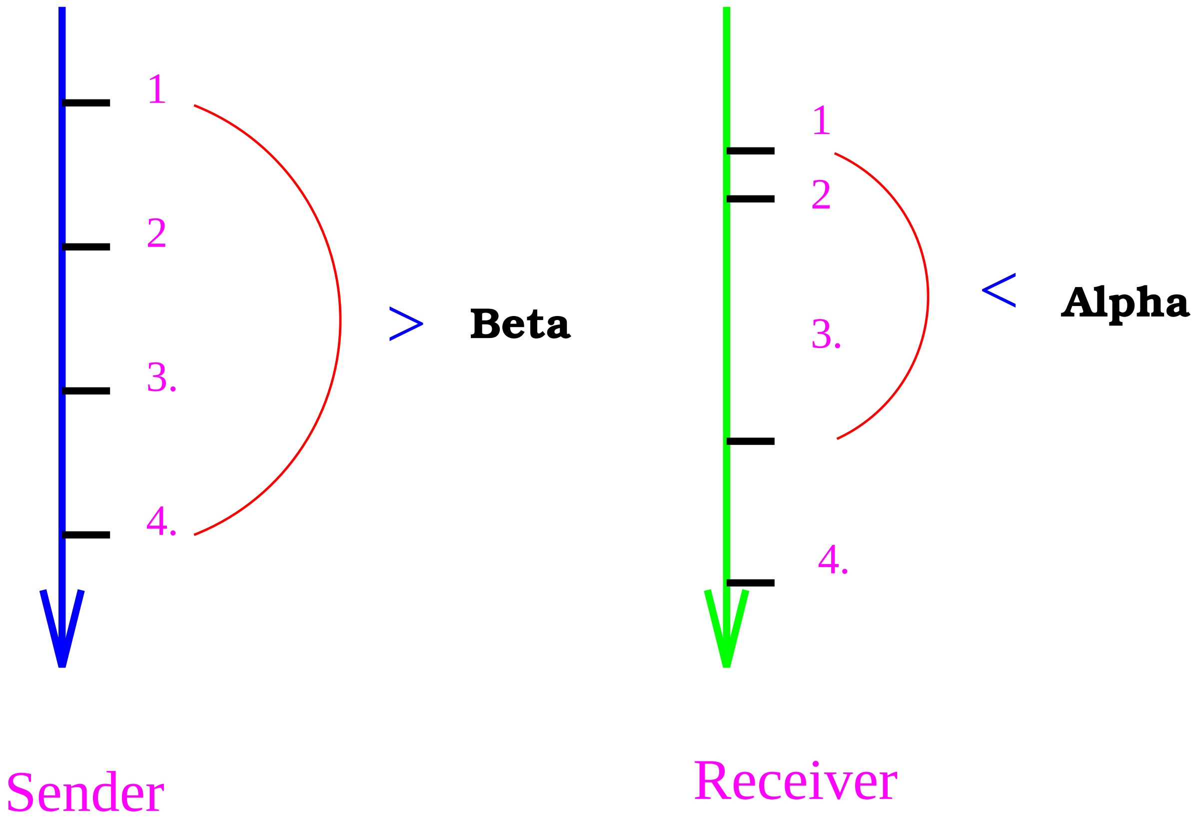

Figure 2: The timing requirements for the Deniable Authentication ZK protocol.

prover that the verifier knows r. We let $x \circ y$ denote the concatenation of x and y, and $x \cdot y$ denote the inner product of x and y modulo 2.

Protocol III: CoZK Deniable Authentication

1. $V \longrightarrow P$: Commitment to $r \in_R \{0,1\}^n$
2. $P \longrightarrow V$: $q_1, q_2, \ldots, q_{2n/3}$, each $q_i \in_R \{0,1\}^n$
3. $V \longrightarrow P$: $E(m \circ r)$, $r \cdot q_1, \ldots, r \cdot q_{2n/3}$
   and NIZK proofs of consistency
   for the inner products
4. $P \longrightarrow V$: 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. See Figure 2. We assume that the adversary is constrained by the (α, β) constraint.

Theorem 4.1 *Protocol III is sound and is concurrent perfect zero-knowledge.*

Proof. (Sketch) The proof of soundness follows from the non-malleability of E, much in the same way as in [17], and the fact that NIZK "proofs" of false statements along with reference strings can be forged that are computationally indistinguishable from good NIZK proofs and random reference strings. The latter fact lets us argue that a cheating prover trying to imitate P is not able to guess r from the responses to its Step 2 queries, because it cannot distinguish valid protocol executions from ones in which the Step 3 responses are wrong and the NIZK proofs and their reference strings have been forged.

The proof of concurrent zero-knowledge is essentially identical to the proof of Theorem 3.2; The proof of Theorem 3.1 can also be adapted; however, in Protocol II the verifier eventually explicitly reveals the value it commits to in Step 1, while in the present case the simulator must obtain this value by knowledge extraction. □

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 IV 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 [33] for definitions and [15, 20] 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.

Protocol IV: On-Line Deniable Authentication
Using Signatures

1. $P \longrightarrow V : C(r_1)$
2. $V \longrightarrow P : r_2$
3. $P \longrightarrow V : auth_{r_1 \oplus r_2}(m)$
4. $P \longrightarrow V : sign(C(r_1) \circ r_2),\ r_1$

Timing Constraints: P delays Step 4 until more than β time has elapsed on P's local clock since r_2 is received in Step 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. The adversary is constrained by the (α, β) constraint. Here the constraint is needed for the security of *both* parties.

Theorem 4.2 *Protocol IV 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. □

Protocol IV *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. □

Remark 4.4 *The use of timing can be partially replaced by employing moderately hard functions.*

The security of Protocol V below is similar to that of Protocol IV; however, it uses a non-malleable encryption function E in place of a signature. Given the elegant and extremely efficient non-malleable encryption scheme of [16], based on the Decisional Diffie-Hellman assumption, this scheme is also efficient.

Protocol V: On-Line Deniable Authentication
Using Non-Malleable PKC

1. $P \longrightarrow V : C(r_1)$
2. $V \longrightarrow P : r_2,\ E(r_2 \circ C(r_1) \circ r_3)$
3. $P \longrightarrow V : auth_{r_1 \oplus r_2}(m)$
4. $P \longrightarrow V : r_3,\ r_1$

Timing Constraints: As in Protocol IV.

5 Recent Work

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 [21]. 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.

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.

[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, Advances in Cryptology–CRYPTO '90, Springer-Verlag,

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

[10] 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.

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

[12] S. Brands and D. Chaum, *Distance-Bounding Protocols* Advances in Cryptology – EUROCRYPT'93, 1993.

[13] D. Chaum and H. van Antwerpen, *Undeniable Signatures*, Advances in Cryptology–CRYPTO '89, G. Brassard (Ed.), Springer-Verlag, pp. 212-216.

[14] 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.

[15] R. Cramer and I. Damgard *New Generation of Secure and Practical RSA-Based Signatures*, Advances in Cryptology–CRYPTO '96. Springer-Verlag, 1996.

[16] R. Cramer and V. Shoup *A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack*, private communication, 1998.

[17] D. Dolev, C. Dwork and M. Naor, *Non-malleable Cryptography*, Preliminary version: Proc. 21st STOC, 1991. Full version: *submitted for publication* (available from the authors).

[18] C. Dwork and M. Naor, *Pricing via Processing -or- Combatting Junk Mail*, Advances in Cryptology – CRYPTO'92, Lecture Notes in Computer Science

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

[20] C. Dwork and M. Naor, *An Efficient Existentially Unforgeable Signature Scheme and its Applications*, Advances in Cryptology – CRYPTO'94, Lecture Notes in Computer Science 839, pp. 234–246; full version to appear, *J. Cryptology*.

[21] C. Dwork and A. Sahai, *Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints*, submitted for publication, 1998.

[22] U. Feige, A. Fiat and A. Shamir, *Zero Knowledge Proofs of Identity*, J. of Cryptology 1 (2), pp 77-94. (Preliminary version in STOC 87).

[23] U. Feige and A. Shamir, *Witness Indistinguishable and Witness Hiding Protocols* Proc. 22nd STOC, 1990, pp. 416–426.

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

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

[26] 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).

[27] 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.

[28] 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.

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

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

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

[32] 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.

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

[34] J. Kilian and E. Petrank, *An Efficient Non-Interactive Zero-Knowledge Proof System for NP with General Assumptions*, Ellectronic Colloquium on Computational Complexity (ECCC)(038), 1995

[35] J. Kilian and E. Petrank, personal communication, 1997

[36] P. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems*, *Proc. CRYPTO'96* pp. 104–113, 1996

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

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

[39] C. Rackoff and D. Simon, *Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Cipherext Attack*, Proc. CRYPTO'91, Springer-Verlag, 1992, pp. 433 – 444

[40] R. L. Rivest, A. Shamir and D. A. Wagner, *Time-lock puzzles and time-release Crypto*, manuscript 1996.

[41] 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.