Responsive Round Complexity and Concurrent Zero-Knowledge

Tzafrir Cohen

Responsive Round Complexity and Concurrent Zero-Knowledge

Research Thesis

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Science

in Computer Sciense

Tzafrir Cohen

Submitted to the Senate of the Technion — Israel Institute of Technology

SHEBAT 5763 HAIFA JANUARY 2003

The Research Thesis Was Done Under The Supervision of Dr. Erez Petrank in the Computer Science Department

Acknowledgment

I wish to thank my advisor, Erez Petrank, for his efforts in the life-cycle of this thesis.

I also with to thank my mother for her encouragement. Dan Kenigsberg and Ido Tal should be thanked for technical help and advise.

THE GENEROUS FINANCIAL HELP OF THE TECHNION IS GRATEFULLY ACKNOWLEDGED

Contents

Abstract **ix**

List of Symbols **xi**

1 Introduction **1**

1.1 Responsive-Round complexity 1

1.2 Concurrent zero knowledge 3

1.2.1 Multi-Party Protocols 5

1.3 Techniques 5

1.4 Contributions 6

1.5 Related work 6

1.6 Terminology 7

1.7 Guide to the paper 7

2 Preliminaries **9**

2.1 Zero-knowledge proofs 9

2.2 Bit commitments 10

2.3 Witness Indistinguishability 11

2.4 Black-box simulation 12

2.5 Concurrent zero-knowledge 13

2.6 The complexity parameters 14

3 Main result **15**

4 Overview of protocol and proof **17**

5 The Zero-knowledge Protocol **19**

6 The simulator **23**

6.1 The rewinding schedule 24

6.2 The Effects of the Rewinds 27

7 Analysis of the Simulator **29**

7.1 Efficiency 29

7.2 Indistinguishability 30

References **35**

Hebrew Abstract **ix**

List of Figures

5.1 The protocol 20

6.1 Description of the Rewinding Schedule of the Simulator 25

6.2 The rewind schedule of one rounds level 3 26

6.3 a simple rewind 28

Abstract

The number of communication rounds is a classic complexity measure for protocols; reducing round complexity is a major goal in protocol design. However, when the communication time is varies between different rounds, and in particular, when one of the parties intentionally delays its messages, the round complexity measure may become meaningless. For example, if one of the rounds takes longer than the rest of the protocol, then it does not matter if the round complexity is bounded by a constant or by a polynomial.

In this paper, we propose a complexity measure called *responsive round complexity*. Loosely speaking, a protocol has responsive round complexity m with respect to Party A, if it makes the following guarantee:

> If A's longest delay in responding to a message in a run of the protocol is t, then, in that run, the overall communication time is at most $m \cdot t$.

This guarantees a quicker response to a party that replies faster whereas a party with a slow connection (or that delays its answer) has a worse guarantee.

We demonstrate the significance of responsive round complexity by presenting a new protocol for concurrent zero-knowledge. The new protocol is a black-box concurrent zero knowledge proof for all languages in $\mathcal{NP}$ with round complexity $\tilde{O}\left(\log^2 n\right)$ but responsive round complexity $\tilde{O}\left(\log n\right)$. While the round complexity of the new protocol is similar to what is known from previous works, its responsive round complexity is a significant improvement: all known concurrent zero-knowledge protocols require $\tilde{O}\left(\log^2 n\right)$ rounds. Furthermore, in light of the known lower bounds for black-box cuncurrent zero-knowledge protocols, the responsive round complexity of this protocol is basically optimal.

List of Symbols

P: A prover in a proof protocol

P^*: Algorithm acting as a prover. Possibly dishonest

V: A verifier in a proof protocol

V^*: Algorithm acting as a verifier. Possibly dishonest

S: A simulator

k: The security parameter

m: The number of round(-pair)s in the preamble of the protocol

β: The time parameter of the protocol. The current timeout.

β_i: The time parameter of level i . $\beta_0 \cdot 2^i$.

Chapter 1

Introduction

In this work, we study a new measure related to the round complexity of protocols. We propose a notion of *responsive round complexity* that properly relates the running time of the protocol with the response time of each of the parties. Finally, we show how to improve state-of-the-art concurrent zero-knowledge protocols with respect to their responsive round complexity, and obtain an (almost) optimal protocol with respect to its responsive round complexity.

1.1 Responsive-Round complexity

Often the communication delay is much longer than the computeation delay. In such a cases it is usually convinient to neglect the computational delay, as long as it is performed in "resonable" time. In such cases the number of rounds can be a good estimation of the time complexity of the protocol. However, a protocol's round complexity is not always directly proportional to its time complexity. The reason is that communication rounds do not always have the same length. Thus, for example, if the length of one of the rounds exceeds the accumulative length of all the other rounds, the round complexity does not tell us anything about the time complexity.

The difference in the length of communication times may be a result of two different reasons. The first one is that the network is unstable and network delays vary during the run of the protocol. The second possible reason is that one of the parties may delay its messages. It is sometimes useful for a party to delay its answer in the protocol until something happens. For example, it may delay

its answer until it obtains information from another source, or it may try to foil timing assumptions made by other parties in the protocol.

We propose a new complexity measure called *responsive round complexity*. Our intention is to relate the overall running time of a party to its response time. By this measure, each party gets a guarantee on the overall communication time, which relates to the longest delay it imposes on the run of the protocol. A party that always responds quickly gets a good guarantee on the overall communication time, and a party that sometimes responds slowly gets a poor guarantee on the overall communication time.

Definition 1.1 Round response time: *We say that the response time of a party A in round i of a specific run σ of a protocol Π is t, if the protocol Π between A and B in run σ tells A to send a message in round i, and t is the length of the time interval starting from the time B sent its message in round $i-1$ and ending at the time B received A's response of round i. (If A is not supposed to send a message in round i, then its response time is 0 for round i.)*

Definition 1.2 Response time: *The response time of party A in run σ of protocol Π is the maximum over all rounds i of A's response time in round i.*

Definition 1.3 Responsive round complexity: *We say that a protocol Π has responsive round complexity m with respect to Party A, if for any possible run σ of Protocol Π, the overall communication time does not exceed $t \cdot m$ where t is A's response time in Run σ of Protocol Π.*

Note that if all rounds are equally long in all runs of the protocol, then the responsive round complexity measure with respect to each of the parties equals the (standard) round complexity measure[1].

Our primary interest in this notion is for cases when a party actually delays its messages to gain something; our goal is to develop protocols in which purposeful delays merely punishes the delayer. However, we note that the guarantee of our protocol also holds for networks with unstable or heterogeneous communication links. Parties will be (unfairly) punished for network delays beyond their control, but this punishment will be roughly proportional to the inherent delays. Thus, someone

[1] Here, we adopt the measure by which a round consists of two messages: one from Party A to Party B and the other is the response of B to A.

with a slow connection will obtain slow service, but will not be starved, as would be the case if a protocol simply timed-out on slow participants.

We demonstrate the usefulness of the new notion by using it for analysing concurrent zero-knowledge protocols and constructing a new protocol with an (almost) optimal responsive round complexity.

1.2 Concurrent zero knowledge

Zero-knowledge interactive proofs as presented by Goldwasser, Micali, and Rackoff [16] are proofs that yield no knowledge but the validity of the proven assertion. These proof systems have shown to be important tools for a variety of cryptographic applications. However, the original definition of zero-knowledge considers security only in a restricted scenario in which the prover and the verifier execute the proof disconnected from the rest of the computing environment.

In recent years, several papers have studied the effect of a modern computing environment on the security of zero-knowledge. In particular, many computers today are connected through networks in which connections are maintained in parallel asynchronous sessions. It would be common to find several connections (such as SSH, a web browser, etc.) running together on a single workstation. Can zero-knowledge protocols be trusted in such an environment?

Zero-knowledge in a concurrent environment was first explored by Feige [11], and by Dwork, Naor, and Sahai [9]. Dwork, Naor and Sahai denoted zero-knowledge protocols that are robust to asynchronous composition *concurrent zero-knowledge* protocols. They observed that several known zero-knowledge proofs, with a straightforward adaptation of their original simulation to the asynchronous environment, may cause the simulator to work exponential time. Thus, it seems that the zero-knowledge property does not necessarily carry over to the asynchronous setting.

Kilian, Petrank, and Rackoff [19] gave the first lower bound for concurrent zero-knowledge, showing that any language that has a 4-round (black-box) concurrent zero-knowledge interactive proof or argument is in BPP. Thus, a large class of known zero-knowledge interactive proofs and arguments for non-trivial languages do not remain zero-knowledge in an asynchronous environment. Rosen [25] has improved this lower bound from 4 rounds to 7. Canetti, Kilian, Petrank and Rosen

[4] have substantially improved the lower bound to $\tilde{\Omega}(\log k)$. [2] The parameter k is the security parameter. A polynomial in k bounds the length of the inputs, the number of proofs that may start concurrently, and the time complexity that the parties spend in the protocol.

On the other hand, Richardson and Kilian [24] exhibited a concurrent zero-knowledge proof for any language in NP. Their protocol requires polynomially many rounds in k. Kilian and Petrank [18] substantially narrow the gap between the upper bound and the lower bounds. Using a different simulator, they provide a tighter security analysis for the Richardson-Kilian protocol, and show that the protocol remains concurrent zero knowledge when run with only $\omega(\log^2 k)$ rounds.

How do these results translate to responsive round complexity? Zero-knowledge is about providing security to the prover. Thus, we expect the prover to follow the protocol and not delay its answers. The verifier is the bad guy, who may choose to deviate from what the protocol dictates in order to get knowledge from the prover. Thus, the verifier may delay its answers, and we would like to investigate how protocols behave in this case. It seems fair to provide quicker service (overall communication time) to verifiers that respond quickly and do not delay their answers. Verifiers that do delay their answers may get an overall slower run of the protocol. Responsive round complexity guarantees that the overall communication time is proportional to the longest delay of the verifier.

Looking at the best known upper bound protocol in [18], it is easy to see that the responsive round complexity with respect to the verifier is equivalent to its round complexity in stable networks. The verifier may simply keep its response time steady, and then the two measures equate. This is the best known protocol with respect to responsive time complexity, and it has responsive time complexity of any function m satisfying $m = \omega(\log^2 k)$.

If we look at the best known lower bound in [4], it provides a specific schedule such that if the protocol does not have enough rounds, no black box simulator can simulate it in this schedule. In the demonstrated schedule each verifier has its own response time, but each of the verifiers does not change its response time during the proof. Thus, the lower bound holds also for responsive round complexity, and we cannot do better than $\tilde{\Omega}(\log k)$.

In this paper we present a new concurrent zero-knowledge proof for all languages in NP that has responsive round complexity m for any function $m = \omega(\log k)$. Namely, the responsive round complexity of this protocol can be set to any function asymptotically larger than $\log k$. Thus, we

[2] The "twiddle" notation neglects multiplicative factors that are polylogarithmic in the main term.

get an algorithm whose responsive round complexity is almost optimal (up to a factor of at most $O(\log k \log\log k)$). This means that any verifier that does not delay its messages (or even just does not change the delay from round to round) is guaranteed a round complexity of $\tilde{O}(\log k)$. Verifiers that do delay their messages get a protocol whose running time is at most $\tilde{O}(\log k)$ times the longest delay they choose to use.

1.2.1 Multi-Party Protocols

Definition 1.3 only referred to two-party protocols (even if several instances may be run concurrently). This is because our current definition of the response-time (1.2) is well-defined only for a case of a "ping-pong" protocol between two parties.

For the general case of a multi-party protocol we need to change the definition of response-time.

Definition 1.4 ***Preceding Messages:*** *For any message M sent in the protocol from host A we denote as messages* preceding *to M the set of messages that were received by A before M was sent. We mark them by B_M*

Definition 1.5 ***Message Response Time:*** *For any message M sent in the protocol from host A we denote as M's* response time *the time between the latest send event of a message in B_M and the receiving of M by its target host.*

Definition 1.6 Response time (multi-party Version): *The response time of a host A in run σ of protocol Π is the maximum over the response time of all the messages A sent in σ.*

We could not

Note that the message response time is measured between two events that are not local to A, and may be on two different hosts. Thus this definition assumes that there is a single clock to the whole system.

1.3 Techniques

One set of previous protocols [24, 18] ignore the timing of the messages and consider only their order; black-box simulatable protocols exist in the general model, though with high round complexity.

Another approach is taken by the protocols of [9, 8]. In this approach, strong restrictions are enforced (or assumed) on the ratio between the slowest and longest response times, simplifying the task of producing a simulation, and allowing for constant-round protocols. One interpretation (and implication) of this restriction is that verifiers with slow response times are treated as malicious verifiers whose responses may be summarily rejected.

Our approach is intermediate between these two approaches. As with the latter approach, we do take response times into consideration, but as with the first approach we place no restriction on these delays. Instead, we monitor the delays and "punish" verifiers with long delays, though in a proportionate fashion. Each verifier has an associated response time that is doubled when the verifier does not respond in time. Thus, there are $O(\log k)$ sets of verifiers, each set containing verifiers responding at around the same time. The prover may delay its answers to each verifier to match its delays with those of the verifier. For each set, we use techniques similar to those in [9, 8] to simulate the conversations with verifiers in this set. We then show that simulating the $O(\log k)$ sets together is still doable in polynomial time.

1.4 Contributions

The first contribution of this work is in proposing the notion of responsive round complexity. We feel that this notion may be useful in settings when one of the parties may gain something from delaying its responses. A guarantee on the responsive round complexity provides a guarantee on the time complexity such that each party "gets what it deserves".

Our second contribution is in providing a concurrent zero-knowledge protocol with almost optimal responsive round complexity. Our design uses the protocol of [24, 18] as a subroutine; its main technical contribution is a method for restarting this subroutine so as to obtain a better protocol in practice.

1.5 Related work

Our notion of responsive-round complexity is of course related to the vast literature on distributed algorithms, and continues the program of studying zero-knowledge in distributed settings.[3] We do

[3] Indeed, we would not be surprised if quite similar definitions have been proposed in this literature.

point out the difference between our notion and the most commonly used distributed model. In the standard distributed model, an adversary can speed up responses in worst case fashion; we require that all parties give a "correct" output by the end of the protocol. In our model, we impose additional requirement on when individual parties finish (give a final output); parties whose responses have been sped up may have to finish long before the end of the protocol as a whole. (Here, the "protocol" is the collective set of interactive proofs)

Several recent works have overcome the difficulty of the asynchronous setting by putting limits on the asynchronisity of the system (timing assumptions) [9, 8, 5] or by making some set-up assumptions on the environment (such as a public key infrastructure) [7, 3].

In an article that was published afhas been published, Prabhakaran, Rosen and Sahai [20] managed to show a protocol that works with $\tilde{O}(\log n)$ in the standard model. They used a different analysis of the simulator of [18] (though with a slightly different protocol). Since the protocol is different they could relax the assumptions they relied on. However the network communication complexity of that protocol has increased.

1.6 Terminology

Some words on the terminology we are using. By zero-knowledge we mean *computational* zero-knowledge, i.e., the distribution output by the simulation is polynomial-time indistinguishable from the distribution of the views of the verifier in the original interaction. Our proof is black-box zero-knowledge. The proof will be perfectly sound, i.e., we will construct an interactive proof, yet it will be possible to run the prover in polynomial time given a witness to the NP assertion that the prover is making.

1.7 Guide to the paper

In Chapter 2 we go over the preliminaries. We state our main result in Chap. 3. We provide an overview on the protocol and proof in Chap. 4. The protocol itself is presented in Chap. 5, the simulator to the protocol is presented in Chap. 6, and the analysis of the simulator is given in Chap. 7.

Chapter 2

Preliminaries

2.1 Zero-knowledge proofs

Let us recall the concept of interactive proofs, as presented by [16]. For formal definitions and motivating discussions the reader is referred to [16].

Definition 2.1 *A protocol between a (computationally unbounded)* prover P *and a (probabilistic polynomial-time)* verifier V *constitutes an* interactive proof *for a language L if there exists a negligible function ε such that*

- **Completeness:** *If $x \in L$ then*

$$\Pr\left[(P,V)(x) \text{ accepts}\right] \geq 1 - \varepsilon(|x|)$$

- **Soundness:** *If $x \notin L$ then for any prover P^**

$$\Pr\left[(P^*,V)(x) \text{ accepts}\right] \leq \varepsilon(|x|)$$

Brassard, Chaum, and Crépeau [2] introduced a modification of interactive proofs, called *arguments*, in which the prover is also polynomial time bounded. Thus, the soundness property is modified to be guaranteed only for probabilistic polynomial time provers P^*.

Let $(P,V)(x)$ denote the random variable that represents V's view of the interaction with P on common input x. The view contains the verifier's random tape as well as the sequence of messages exchanged between the parties.

We briefly recall the definition of black-box zero-knowledge [16, 23, 14, 15]. The reader is referred to [15] for more details and motivation.

Definition 2.2 *A protocol (P, V) is* computational zero-knowledge *(resp.,* statistical zero-knowledge*) over a language L, if there exists an oracle polynomial time machine S (simulator) such that for any polynomial time verifier V^* and for every $x \in L$, the distribution of the random variable $S^{V^*}(x)$ is polynomially indistinguishable from the distribution of the random variable $(P, V^*)(x)$ (resp., the statistical difference between $M(x)$ and $(P, V)(x)$ is a negligible function in $|x|$).*

In this paper, we concentrate on black-box computational zero-knowledge. In the sequel we will say *zero-knowledge* meaning *black-box computational zero-knowledge.*

2.2 Bit commitments

We include a short and informal presentation of commitment schemes. For more details and motivation, see [12]. A commitment scheme involves two parties: The *sender* and the *receiver.* These two parties are involved in a protocol which contains two phases. In the first phase the sender commits to a bit, and in the second phase it reveals it. A useful intuition to keep in mind is the "envelope implementation" of bit commitment. In this implementation, the sender writes a bit on a piece of paper, puts it in an envelope and gives the envelope to the receiver. In a second (later) phase, the *reveal* phase, the receiver opens the envelope to discover the bit that was committed on. In the actual digital protocol, we cannot use envelopes, but the goal of the cryptographic machinery used, is to simulate this process.

More formally, a commitment scheme consists of two phases. First comes the *commit* phase and then we have the *reveal* phase. We make two security requirements which (loosely speaking) are:

Secrecy: At the end of the *commit phase*, the receiver has no knowledge about the value committed upon.

Binding property: It is infeasible for the sender to pass the commit phase successfully and still have two different values which it may reveal successfully in the reveal phase.

Various implementations of commitment schemes are known, each has its advantages in terms of

security (i.e., binding for the receiver and secrecy for the receiver), the assumed power of the two parties etc.

Two-round commitment schemes with perfect secrecy can be constructed from any collection of claw-free permutations; see [12]. It is shown in [2] how to commit to bits with statistical security, based on the intractability of certain number-theoretic problems. Dåmgard, Pedersen and Pfitzmann [6] give a protocol for efficiently committing to and revealing strings of bits with statistical security, relying only on the existence of collision-intractable hash functions. This scheme is quite practical and we adopt it for the verifiers in our protocol. For the prover, we use a commitment scheme whose binding is information theoretic and security is computational. Such schemes can be constructed from any one-way function, see [21]. For simplicity, we simply speak of committing to and revealing bits when referring to the protocols of [6] for the verifier and [21] for the prover. We will need to use the properties of the commitment schemes in the concurrent setting.

Theorem 2.3 *The security of the bit commitments in [21] and [22] holds also in the concurrent setting.*

Proof: By definition, the binding property must be robust to asynchronous composition. Otherwise, the committer may play a mental game in which his real stand-alone commitment is part of an asynchronous game which he simulates, and then defeat the binding property in the normal stand-alone world.

As for the secrecy, a similar argument may be more complicated, since the receiver cannot simulate the behaviour of the committer. Specifically, the committer has some information that the receiver does not have: the value of the committed string, which may be used in the other commitments. However, in our proof, the committer commits on uniformly chosen random strings. (And on nothing else.) Thus, if the committer follows the protocol, then the receiver is able to simulate the rest of the environment and the above argument holds for secrecy as well. ■

2.3 Witness Indistinguishability

Witness indistinguishable proofs were presented in [10]. The motivation was to provide a cryptographic mechanism whose notion of security is similar though weaker than zero-knowledge, it is

meaningful and useful for cryptographic protocols, and the security is preserved in an asynchronous composition. A witness indistinguishable proof is a proof for a language in $\mathcal{NP}$ such that the prover is using some witness to convince the verifier that the input is in the language, yet, the view of the verifier in case the prover uses witness w_1 or witness w_2 is polynomial time indistinguishable. Thus, the verifier gets no knowledge on which witness was used in the proof. The formal definition follows. For further discussion and motivation the reader is referred to [10].

We say that a relation R is polynomial time if there exists a machine that given (x, w) works in polynomial time in $|x|$ and determines whether $(x, y) \in R$. For any $\mathcal{NP}$ language there exists a polynomial time relation R_L such that L can be described as $L = \{x : \exists y, \quad R_L(x, y)\}$.

Definition 2.4 *A proof system (P, V) is witness indistinguishable over a polynomial time relation R is for any V', any large enough x, any w_1, w_2 such that $(x, w_1) \in R$ and $(x, w_2) \in R$, and for any auxiliary input y for V', the view of V' in the interaction with $P(x, w_1)$ is polynomially indistinguishable from the view of V' in the interaction with $P(x, w_2)$.*

It is shown in [10] that witness indistinguishability is preserved with asynchronous composition of proofs. We use a constant round zero-knowledge proof for the languages in $\mathcal{NP}$ such as in [13]. Any zero-knowledge interactive proof is also witness indistinguishable. Furthermore, any witness indistinguishable proof is a concurrent witness indistinguishable proof. Thus, we obtain a constant round concurrent witness indistinguishable proof that we use in our interactive proof.

2.4 Black-box simulation

The initial definition of zero-knowledge [15] requires that for any probabilistic polynomial time verifier $\hat{V}$, a simulator $S_{\hat{V}}$ exists that simulates $\hat{V}$'s view. Oren [23] proposes a seemingly stronger, "better behaved" notion of zero-knowledge, known as *black-box* zero-knowledge. The basic idea behind black box zero-knowledge is that instead of having a new simulator $S_{\hat{V}}$ for each possible verifier, we have a single probabilistic polynomial time simulator S that interacts with each possible $\hat{V}$. Furthermore, S is not allowed to examine the internals of $\hat{V}$, but must simply look at $\hat{V}$'s input/output behaviour. That is, it can have conversations with $\hat{V}$ and use these conversations to generate a simulation of $\hat{V}$'s view that is computationally indistinguishable from $\hat{V}$'s view of its interaction with P.

At first glance, the limitations on S may seem to force S to be as powerful as a prover. However, S has important advantages over a prover P, allowing it to perform simulations in probabilistic polynomial time. First, it may set $\hat{V}$'s coin tosses as it wishes, and even run $\hat{V}$ on different sets of coin tosses. More importantly, S may conceptually "back up" $\hat{V}$ to an earlier point in the conversation, and then send different messages. This ability derives from S's control of $\hat{V}$'s coin tosses; since $\hat{V}$ otherwise operates deterministically, S can rerun it from the beginning, exploring different directions of the conversation by trying various messages.

In our setting, the clock plays a central role in the transcript of the interaction. Thus, we would like to specifically require that the simulator may rewind the clock of the verifier when using it as a black box. This means that the simulator has the ability to restart the verifier's clock as well as reset its machine state. This requirement is actually implicit in all previous work on black-box zero-knowledge. Note that if the verifier can be rewound without its clock being adjusted, then the verifier can tell it is rewound and behave differently in the second run of the rewind.

Indeed, all known proofs of zero-knowledge construct black-box simulations. There is no way known to make use of a verifier's internal state, nor to customise simulators based on the description of $\hat{V}$ other than by using it as a black box.[1] Thus, given the current state of the art, an impossibility result for black-box zero-knowledge seems to preclude a positive result for the older definitions of zero-knowledge.

2.5 Concurrent zero-knowledge

Following [9], we consider a setting in which a polynomial time adversary controls many verifiers simultaneously. The adversary $\mathcal{A}$ takes as input a partial conversation transcript of a prover interacting with several verifiers concurrently, where the transcript includes the local times on the prover's clock when each message was sent or received by the prover. The output of $\mathcal{A}$ will be a tuples of the form (V, α, t), indicating that P receives message α from a verifier V at time t on $P's$ local clock. The adversary may either output a new tuple as above, or wait for P to output its next message to one of the verifiers. The time that is written by the adversary in the tuple, must be greater than all times previously used in the system (by messages sent to P or by P). The view of

[1] As one slight exception, [17] proves security against space-bounded verifiers by considering the internal state of the verifiers. However, these techniques do not seem applicable to more standard classes of verifiers.

the adversary on input x in such an interaction (including all messages and times, and the verifiers random tapes) is denoted $(P, \mathcal{A})(x)$.

Definition 2.5 *We say that a proof or argument system (P, V) for a language L is (computational)* concurrent zero-knowledge *if there exists a probabilistic polynomial time oracle machine S (the simulator) such that for any probabilistic polynomial time adversary $\mathcal{A}$, the distributions $(P, A)(x)$ and $S^{\mathcal{A}}(x)$ are computational indistinguishable over the strings that belong to the language L.*

In what follows, we will usually refer to the adversary $\mathcal{A}$ as the *adversarial verifier* V^* or just the *verifier* V^*. All these terms mean the same.

In our setting, the simulator will simulate a predetermined time interval which is polynomial in k. We assume that while rewinding the verifier, the simulator may also set its clock to the required rewound time.

2.6 The complexity parameters

In this paper, we simplify the discussion by using a single security parameter k. Our proof has (in worst case) $\omega(\log^2 k)$ rounds and it has responsive round complexity $\omega(\log k)$. The zero-knowledge simulation is guaranteed for a polynomial (in k) number of concurrent proofs. Also, the running time of the protocol is polynomial in k. We will measure time by the smallest time units that are relevant in this setting. For example, one may think of the time unit as the minimal time a round in the protocol may take. But we may also use a much smaller time unit: the time of a computer cycle. In any of these time units, it holds that the running time of the protocol is polynomial (in k).

Chapter 3

Main result

Our main result is the existence of black-box concurrent zero-knowledge interactive proof for all languages in $\mathcal{NP}$ with responsive round complexity m for any m satisfying $m = \omega(\log k)$. We state this explicitly in the following theorem.

Theorem 3.1 *Assume there exist secure two-round commitment schemes with statistical secrecy and secure two-round commitment schemes with statistical binding (such schemes follow from the existence of a family claw-free permutation pairs). Let k be a complexity parameter bounding the size of the input. The verifier is polynomial time in k, and the concurrent proof may contain a polynomial (in k) number of proofs concurrently. Then there exists a black-box concurrent zero-knowledge interactive proof for all languages in $\mathcal{NP}$, with:*

- *responsive round complexity $m(k)$, for any function $m(k)$ satisfying $m(k) = \omega(\log k)$, and*
- *a worst case round complexity of $m(k) \cdot \log k$.*

Chapter 4

Overview of protocol and proof

We start with the protocol in [24, 18]. We choose the following parameters for this protocol: the preamble consists of m rounds for $m = \omega(\log k)$ (recall that a round consists of a message sent from the prover to the verifier followed by a response of the verifier). The body of the proof consists of a low error, constant round, auxiliary-input witness-indistinguishable interactive proof for $\mathcal{NP}$ in which the prover can be efficiently given the witness to the proven assertion. We use the zero-knowledge protocol of [13] for that.

When a new copy of the protocol is initiated by the verifier, the verifier in the new protocol is associated with a response time which is initially the minimal possible response time, say the time of a computer cycle [1]. When the verifier fails to respond within this time, the time associated with this verifier is doubled and the verifier is notified that it must start again with the doubled time. In this case we say that the verifier has been reset and has gone one level up. This may happen at most $O(\log k)$ times since at some such level the response time becomes greater than the running time of the adversary, or bigger than the time interval that has to be simulated. The verifiers may be viewed as working in levels of responsiveness. Level i contains all verifiers with response time at most $\beta_i = 2^i$ and greater than β_{i-1}. The prover treats each verifier independently in light of its associated response time or level. For each verifier, the prover delays its answer according to its associated delay β in a manner yet to be discussed.

[1] The initial time must be the minimal time unit, which is indeed a limitation of the protocol, as the parties cannot start from an estimated delay. This is essential to the responsive-round complexity analysis alone, though. The other properties of the protocol will hold even if each protocol may begin with a longer delay

The completeness and soundness of the interactive proof hold as in [24, 18]. The worst case number of rounds for this protocol happens when the verifier goes through m steps in each level and then delays its last message and is reset while going up to the next level. This yields $m \cdot O(\log k)$ rounds in the worst case. But since each level takes double the time of the previous level, the overall interaction time is dominated by the time of the highest level interaction, and is $O(m)$.

It remains to prove that the interactive proof is zero-knowledge. The delays imposed by the prover are similar to those suggested in [9]. Thus, simulating all protocols at the same level becomes possible in a way similar to that in [9]. See also Fig. 6.3 later on. The delay imposed by the prover are not more than twice β_i for a verifier in level i and thus, do not increase the protocol time too much. It remains to show that rewinding protocols at higher levels do not force too many rewinding at lower levels. This is obtained with some care in the setting of the prover delays and by the fact that there are at most a logarithmic number of levels.

The reason we need a logarithmic number of rounds and cannot do with a constant number of rounds for each level as in [9] is the relation between the various levels. We allow ourselves one rewind only to any interval we wish to rewind. Any other constant will do, but rewinding a super-constant number of times (or polynomial as in [9]) will make the overall simulation time super-polynomial. Note also that this is an inherent problem since the lower bound in [4] uses verifiers that in each specific copy of the proof do not modify their response time. Thus the lower bound holds also for responsive round complexity and we cannot do with asymptotically less than $\log k / \log\log(k)$ responsive round complexity.

Chapter 5

The Zero-knowledge Protocol

We start by presenting the protocol. It consists of a preamble of m rounds where m is any function satisfying $m = \omega(\log k)$ and a body consisting of a (not concurrent) constant round zero-knowledge proof. If this were the full picture, we would get that the overall number of rounds is dominated by m and is thus almost logarithmic. However, we sometimes let the prover say "RESET". This happens only during the preamble, and is caused by a long delay in the verifier response. When such a delay occurs, the protocol starts from the beginning with a delay parameter doubled. At this point we say that the proof has gone up one *level*. Generally a proof is at level i if it has gone through i RESETs.

To see that the overall round complexity is $m \cdot O(\log k)$ it is enough to note that the maximum number of resets is logarithmic. This is true since the delay can only be doubled a logarithmic number of times. The logarithm is in the length of the simulated period. We denote this length by Δ and measure it in units of β_0, i.e., the time of a computer cycle. In Figure 5.1 we describe the protocol. This is the protocol presented in [24, 18] enhanced with time monitoring and possible resets. All commitments from the verifier to the prover are statistically secret and all commitments from the prover to the verifier are statistically binding.

Theorem 5.1 *If the zero-knowledge proof used in the body of the protocol has completeness error ϵ_c and soundness error ϵ_s then our interactive proof as in Fig. 5.1 has completeness error ϵ_c and soundness error at most $\varepsilon_s + \varepsilon$ for some negligible fraction ε.*

Step V-0:	V →P: V Selects m strings, $v_1, \ldots, v_m \in \{0,1\}^n$ uniformly and independently at random, and send Commit $(v_1), \ldots,$ Commit (v_m) to the prover.
Step P-1:	P →V: Send Commit (p_1) exactly T after the Step V-0 message was sent.
Step V-1:	V →P: Reveal v_1.
⋮	
Step P-j:	P →V: If V's message from Step V-$(j-1)$ was received more than T time units after P's message from Step P-$(j-1)$ was sent then goto RESET. Else, send Commit (p_j) exactly $2T$ time units after P's round $(j-1)$ message was sent.
Step V-j:	V →P: Reveal v_j.
⋮	
Step P-m:	V →P: If V's message from Step V-$(m-1)$ was received more than T time units after P's message from Step P-$(m-1)$ was sent then goto RESET. Else, send Commit (p_j) exactly $2T$ time units after P's round $(j-1)$.
Step V-m:	V →P: Reveal v_m.
Proof body:	P waits T time units and then proves to V in zero-knowledge that $x \in L$ or that $\exists i$, $1 \leq i \leq m$, such that $p_i = v_i$. (No delays or time monitoring is used during the course of this proof.)
End of proof	
RESET:	
P →V:	A reset message with parameter $2T$. Both P and V continue by setting $T = 2T$ and starting the protocol from Step V-0 again.

Figure 5.1: The protocol
איור 5.1: הפרוטוקול

Proof: Clearly, the completeness error cannot increase. As for the soundness, the prover may gain extra strength by managing to set $p_i = v_i$ for some $1 \leq i \leq m$. However, since the verifier is using statistically hiding commitment scheme this may happen with negligible probability only, and we are done. ■

Lemma 5.2 *The protocol has responsive round complexity* $5m$.

Proof: We show that it holds for the preamble. The additional constant number of rounds in the body of the proof cannot increase the responsive round complexity since the prover answers with no delays at that stage of the protocol.

Consider a proof that ended the preamble at level ℓ, i.e., had ℓ resets. (We will discuss later the case that the preamble has not ended at all within the time Δ.) If $\ell = 0$ then the proof had m rounds, the length of each equals the minimum possible response time, that was actually matched

by the verifier. Otherwise, we have $\ell \neq 0$. The proof was last reset at level $\ell - 1$ which means that the verifier did not respond within time

$$\beta_{\ell-1} = 2^{\ell-1} = \frac{1}{2}\beta_\ell.$$

Thus the response-time of the verifier is at least $\beta_{\ell-1}$. We now compute the overall communication time and show that it is smaller than $5m \cdot \beta_{\ell-1}$.

At each of the levels $i = 1, 2, \ldots, \ell - 1$ the protocol ran for at most m rounds. At level ℓ we assume it finished the preamble and thus had m rounds. Summing over all the communication times during the preamble we get that the communication time is bounded by

$$\sum_{i=1}^{\ell} m \cdot \beta_i \;= m \cdot \sum_{i=1}^{\ell} 2^i \;\leq\; m \cdot 2^{i+1} \;=\; 4 \cdot m \cdot \beta_{\ell-1}.$$

We bound the additional communication time of the proof body by $m\beta_{\ell-1}$. This is correct for the constant round body if the verifier does not pose a delay longer than $\beta_{\ell-1}$; if it does, the responsive round complexity may only decrease.

Last, we deal with the case that the verifier does not finish. We assume that the simulation time Δ is much larger than the running time of the adversarial verifier. Thus, a particular verifier that has not yet responded will never respond and its responsive round complexity is much better than $5m$. ■

We next show that the protocol is concurrent zero-knowledge, by presenting a simulator for the concurrent interaction.

Chapter 6

The simulator

We present a black box simulation of the above protocol. We assume the worst, i.e., that there is one adversary that controls all verifiers (whose number is polynomial in k). This adversary deviates from the protocol as it wishes and is limited only by being a polynomial time machine. The simulator interacts with this adversary (or with these verifiers) and its goal is to produce a transcript distribution which is indistinguishable from the real interaction between the adversary and the original prover P. Note that each message in the transcript is associated with a time telling when it is produced after the beginning of the interaction.

The simulator simulates the body of the proof simply by playing the real prover. The reason it may do that is that it rewinds each of the verifiers so that it manages to get a round i in which $p_i = v_i$. After that we say that this particular copy of the proof has been "solved", or that this particular verifier has been *neutralised*. Our goal is to ensure that there will be enough rewinding so that all proofs will be solved, while taking care that the rewinding does not exceed polynomial time.

The difficulty in the construction and in describing the simulator lies in the rewinding schedule. Other than that the operation of the simulator is quite simple. The simulator runs the adversary on a randomly chosen random string while performing all rewinds in the rewind schedule. The simulator breaks the entire sequence of time steps into sections. Each section is simulated twice by the simulator. The first run is used to obtain information, and the second run is used to produce the actual output transcript. During simulation of each such section, the simulator recursively divides

it into smaller subsections.

During the first time a section is simulated, the simulator records the strings revealed by the verifiers during this run. Then, while running the second run of the rewind, the simulator solves all proofs that may be solved by setting p_i to equal v_i for known values of v_i's. The second run of the section is used to produce the transcript obtained thus far. When a body of a proof arrives, if the proof has been solved, then the simulator acts as the prover while proving the existence of i such that $p_i = v_i$ (the simulator has a witness to this fact). If the proof has not been solved, the simulation aborts and declares failure.

We will show that the probability that any of the proofs remains unsolved is negligible. Thus, the simulator rarely fails. When it does not fail, its output will be indistinguishable from the real interaction. One difference between the simulated transcripts and the real ones is in the preambles: in the simulation there is an i with $p_i = v_i$ (in fact, all, or at least most, of the rounds, should have $p_i = v_i$). But by the secrecy of the commitment schemes this difference cannot be detected by a polynomial-time bounded machine. Note that these strings are never revealed, avoiding difficulties arising when partial subsets are revealed. The second difference is in the witness used in the bodies of the proofs. However, a zero-knowledge proof is witness indistinguishable. This property is preserved in a concurrent setting and is thus indistinguishable by a polynomial time distinguisher.

It remains to show that there exists a rewinding schedule by which the simulator is efficient and still all proofs are solved with overwhelming probability.

6.1 The rewinding schedule

The schedule of the rewinds is given as a pseudo-code in Fig. 6.1 and is illustrated in Fig. 6.2.

The X axis represents the time (as viewed by the verifiers or listed in the output transcript produced by the simulator), and the numbers in the graph represent the X coordinate (=the time) of an event. The Y axis represents the order of events of the simulator itself. The advances of the simulation are shown as thick arrows, whereas the rewinds are shown as thin backward arrow.

Thus ($3 \leftarrow 5$) near a rewind arrow means that the simulator has rewound the verifiers back from time 5 to time 3, and the numbers `3 4 5 6` on an advancing arrow indicate that the simulation has progressed through times 3–6.

```
0:   // Recall that Δ is the overall simulation time interval.
1:   top_level = log(Δ)
2:
3:   // This is a recursive algorithm. Top-level call follows:
4:   simul(0, Δ, top_level)
5:   output transcript
6:
7:   // Definition of recursive function:
8:   simul(location, length, level)
9:     β = 2^level
10:    if(level < 1)
11:      // here comes the simulation of interaction with V* for time interval length
12:      return
13:    else
14:      // recursively run lower-level simulations
15:      for i = 0 to length/β - 1
16:        simul(location + i·β, β,level-1)
17:        simul(location + (i+1)·β, β,level-1)
18:        rewind(location + i·β)
19:        simul(location + i·β, β, level-1)
20:      end for
21:    end if
```

Figure 6.1: Description of the Rewinding Schedule of the Simulator
איור 6.1: תיאור של תזמון ההחזרות בסימולטור

In the example of Fig. 6.2 the top-level run has just one recursive section. At the top level this is not always the case, but in any other level the recursion is invoked exactly twice. The top level of this run is $\log \Delta$ (all logarithms are base 2), where Δ is the length of the interval we simulate. In the example $\Delta = 4$ and the top level is 2. The first section starts at the beginning and ends when the simulator advances from 1 to 2 after the after the seventh rewind (the second $(1 \leftarrow 3)$). The second section begins in the advancement from 2 to 4 and ends at the end of the simulation.

During the run of the top-level sections there are also rewinds of lower levels. In this example there is only one lower level: level 1. For each rewind of level ℓ there are 6 rewinds of level $\ell - 1$ (4 before the actual level-ℓ rewind, and two after it). Thus before the first level-2 rewind $((0 \leftarrow 4))$ there are 4 level-1 rewinds $((0 \leftarrow 2)$, $(1 \leftarrow 3)$, $(2 \leftarrow 4)$ and $(3 \leftarrow 5))$ and more two after it $((0 \leftarrow 2)$ and $(1 \leftarrow 3))$.

Note that the rewinding schedule does not depend on the schedule of proofs as determined by

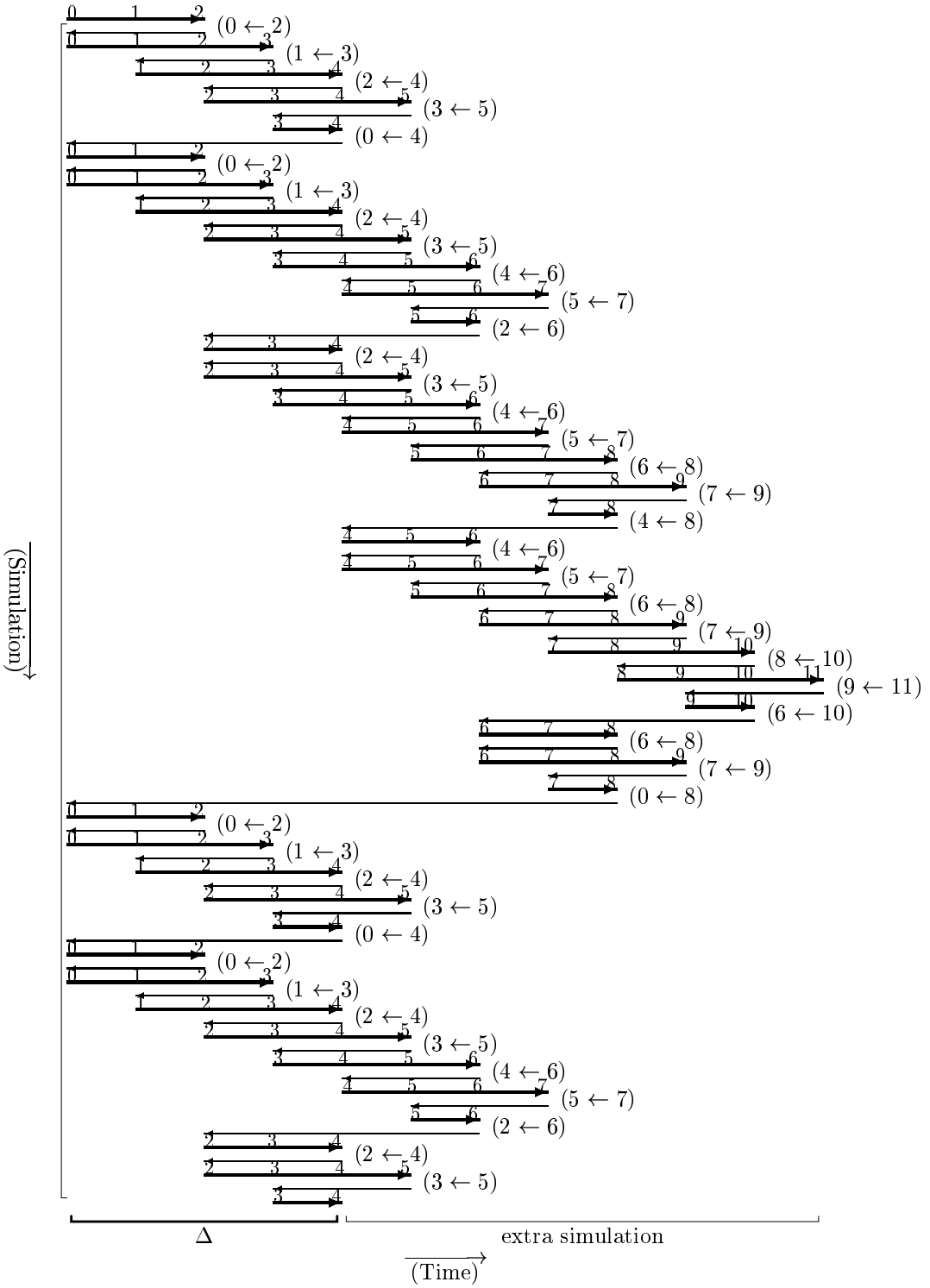

Figure 6.2: The rewind schedule of one rounds level 3

איור 6.2: תזמון ההחזרות בריצה של סיבוב אחד מרמה 3

the adversarial verifier. It may be the case that no proof ran and the simulator would still behave the same. The rewinding schedule depends on the time only.[1]

6.2 The Effects of the Rewinds

Each proof may start its i-th level run in an arbitrary point in time. However, by the delay of β_i imposed by the prover, they all have the same look during the preamble: The prover sends a message, then the verifier responds within time β_i and then the prover sends its next message exactly $2\beta_i$ time units after its previous message. Thus, the time between two prover messages is always $2\beta_i$ and the response time of the verifier is less than β_i.

A *rewind* operation at level i that makes the simulator run the interval $(T, T + 2\beta_i)$ twice is meant to solve all proofs of level i in which the verifier sent a preamble message "`Reveal` v_ℓ" (for some $1 < \ell \leq m$) in response to a prover message that was sent in-between the times T and $T + \beta_i$. Note that in this case the verifier must respond before $T + 2\beta_i$ and thus simulator has learned the value of v_ℓ. Since p_ℓ is still within the rewind. The simulator may modify the commitment on p_ℓ in the second run of the rewind and commit on $p_\ell = v_\ell$. For example, the rewind $(0 \leftarrow 4)$ may solve proofs that run in the top level and whose prover has sent a message between the times 0 and 2.

figure 6.3 shows a short phase of the preamble between the prover and a number of verifiers in of the same levels (level i: with delay β_i). That figure demonstrates that a rewind every β_i will indeed suffice for a simulator to rewind each verifier on each round: the rewind $(T + 2\beta_i \leftarrow T)$ is is effective for all the rounds for which the prover sends its message between T and $T + \beta_i$.

See Fig. 6.3

Had this always worked, we wouldn't need so many preamble rounds. A couple of them would have been enough. However, here is what may go wrong: the verifier may delay its answer in the first run of the rewind interval, thus getting a reset message from the prover (simulator), yet, in the second run, provide an answer in time. In this case, the simulator would not know the value of v_ℓ in the second run since it was not exposed in the first run. The second run is the one that prevails and written to the final transcript. Thus, solving the proof in this round of the preamble fails in

[1] We remark that one may obtain better efficiency by checking if the rewind is helpful to the simulation and avoid rewinding if it's not. Even if one does not try to check the messages, a scrutiny of the schedule may lead to other improvements. However all we care about is that the simulation is polynomial time and this is guaranteed by our simple non-optimised simulation procedure.

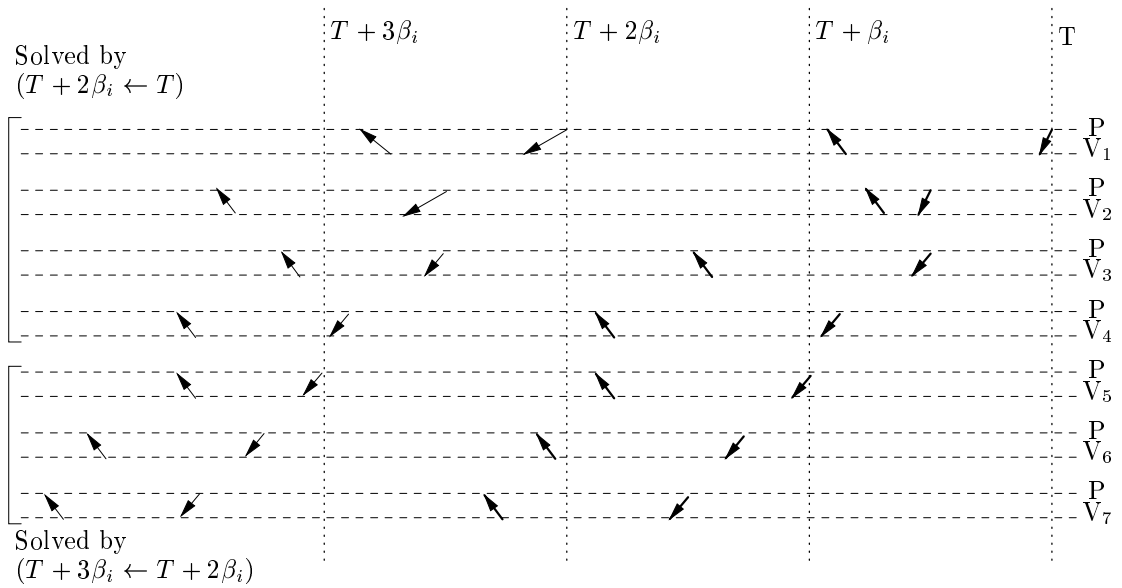

Figure 6.3: a simple rewind
איור 6.3: הדגמת החזרה אחת

This figure shows how the simulator's rewinds "cover" all the session with the same delay (β_i, in this example) with constant number of rewinds. V_1, V_2, V_3 and V_4 were sent a preamble-round message between times T and $T + \beta_i$. A response from each must be received after at most β_i. Thus after $2\beta_2$ all of them have either replied or timed-out.

this case. In Chap. 7 below, we argue that this happens with constant probability in each round and with negligible probability in all the m rounds of the preamble. Note that setting $p_\ell = v_\ell$ in any one of the rounds suffices to solve the proof.

It remains to analyse the probability that the simulator succeeds in solving each of the proofs (before getting to the proof body) and to verify that the rewinding schedule results in a polynomial time simulator. This analysis is provided in the next chapter.

Chapter 7

Analysis of the Simulator

7.1 Efficiency

We start by showing that the rewinding schedule of Section 6.1 results in a polynomial time simulator. Note that each step of the simulation, i.e., committing on strings, revealing them, and playing the prover in the proof-body are all polynomial time. Thus, if the rewinding is polynomial time, we get that the whole algorithm is efficient. We will actually show that the number of rewinds is polynomial. Since each rewind time is polynomial this is enough.

Lemma 7.1 *The overall number of rewinds during the simulation run on time interval Δ is at most Δ^3.*

Proof: We use the recursive description of the rewinding schedule as in Fig. 6.1. Consider a run of a time interval t at level ℓ. Using the notation of Fig. 6.1 this is a run of $simul(location, t, \ell)$. Note that the number of rewinds is independent of the location. Thus, we denote the number of rewinds in this run by $X(t, \ell)$. In a run of t time units at level ℓ there are $\frac{t}{\beta_\ell}$ iterations of the main loop of the **simul** procedure. In each iteration there is a level-ℓ rewind and 3 calls to **simul**$(\cdot, \beta_\ell, \ell - 1)$ are performed (recall that $\beta_\ell = 2^\ell$). Thus,

$$X(t, \ell) \;\leq\; \frac{t}{\beta_\ell} \cdot (1 + 3 \cdot X(2\beta_{\ell-1}, \ell - 1)) \,. \tag{7.1}$$

And this recursion inequality gives the bound:

$$\begin{aligned}
X(t,\ell) &\leq \frac{t}{\beta_\ell} \cdot (1 + 3 \cdot X(2\beta_\ell, \ell - 1)) && (7.2)\\
&\leq \frac{t}{\beta_\ell} \cdot \left(1 + 3 \cdot \left(\frac{\beta_\ell}{\beta_{\ell-1}} \cdot (1 + 3X \cdot (\beta_{\ell-1}, \ell - 2))\right)\right) && (7.3)\\
&= \frac{t}{\beta_\ell} \cdot (1 + 3 \cdot (2 \cdot (1 + 3X \cdot (\beta_{\ell-1}, \ell - 2)))) && (7.4)\\
&= \frac{t}{\beta_\ell} \cdot (1 + 6 \cdot (1 + 3X \cdot (\beta_{\ell-1}, \ell - 2))) && (7.5)\\
&\leq \ldots && (7.6)\\
&\leq \frac{t}{\beta_\ell} \cdot (1 + 6 \cdot (1 + 6 \cdot \ldots (1 + 3 \cdot X(\beta_2, 1)) \ldots)) && (7.7)\\
&= \frac{t}{\beta_\ell} \cdot (1 + 6 \cdot (1 + 6 \cdot \ldots (1 + 3 \cdot 2) \ldots)) && (7.8)\\
&\leq \frac{t}{\beta_\ell} \cdot 7^{\ell-1} && (7.9)
\end{aligned}$$

And at the top level: $t = \Delta, \ell = \log \Delta$

$$\frac{t}{\beta_\ell} \cdot 7^{\ell-1} = \frac{\Delta}{2^{\log \Delta}} \cdot 7^{\log \Delta} = 7^{\log \Delta} \leq \Delta^3 \quad (7.10)$$

as required. ■

7.2 Indistinguishability

We proceed by showing that no polynomial-time algorithm can distinguish the output of the simulator from V^*'s view of its interaction with the original prover, P.

Proof: We discuss separately the preambles and the proof bodies. The difference in the *preambles* is that in the simulation one or more of the rounds has $p_i = v_i$. In the real interaction, this may only happen with negligable probability.

The difference in the *bodies* is that the simulator always proves that "$\exists i$ such that $p_i = v_i$" whereas the original prover (almost) always proves that $x \in L$.

The prover never opens its commitments on the p_i's. By the secrecy of the prover's commitment scheme, a polynomial time distinguisher has negligble probability tell between preambles generated by the simulators and real preambles.

When the simulator has reached the main part it has a witness to its claim ("$\exists i$ such that $p_i = v_i$") with very high probability, according to Lemma 7.2. The proof bodies are witness indistinguishable.

By [10] this property holds also in the concurrent setting. Thus, an efficient distinguisher cannot tell between using a witness to "$\exists i$ such that $p_i = v_i$" and using a witness to "$x \in L$" and we are done.

Assume that in the course of the simulation there are at-most $r_1 = p(k)$ instances of the protocol. According to lemma 7.1 this means that there are at-most $\Delta^3 \cdot m \cdot r_1$ rounds. Thus there have been at-most $\Delta^3 \cdot m \cdot r_1$ bit-commitments by the prover. ■

Lemma 7.2 *The simulator may fail to solve one of the proofs only with negligible probability.*

Proof: We first argue (Lemma 7.3) that a rewind will solve a rewound proof round with a probability of at least $\frac{2}{3}$. We then argue (in Lemma 7.8) that for each proof, each round of its preamble that appears in its final level is rewound. Since all rewinds are rewound independently, and since solving the proof in one of them is enough, and since there are $m = \omega(\log k)$ such rewinds before the body of the proof, we get that the proof remains unsolved with probability $\left(\frac{1}{3}\right)^m$, which is negligible. Any of the proofs may be run a polynomial number of times by the simulator (since intervals are rewound) and there are a polynomial number of proofs. By the summation bound, the probability that any of these proofs is not solved by the end of the preamble remains negligible. ■

The remaining lemmas:

Lemma 7.3 *If in a proof Π at level ℓ the prover's message for round i is sent at time T, then a rewind ($T' \leftarrow T' + 2\beta_\ell$), where $0 \leq T' - T < \beta_\ell$, solves Π in this rewind with probability at least $\frac{2}{3}$.*

Proof: If the verifier answers in time during both runs of the rewind then the proof is solved: the first run reveals the value v_j (for round j of the proof) and in the second run of the rewind the prover may commit on a modified $p_j = v_j$. By the secrecy of the verifier's commitment scheme, the verifier cannot modify v_j, except with negligible probability, since it is committed to this value as of the first round of the proof Π.

If the verifier does not answer in the second run of the rewind, then it is actually reset into level $\ell + 1$ and this proof does not need to be solved at level ℓ. The only bad case is when the verifier delays its answer in the first run, but does not delay it in the second run of the rewind. In this case, the simulator does not learn the value of v_j in the first run and thus, cannot set $p_j = v_j$ in the second run.

What is the probability of this bad incident? Since the prover commits to p_j, the verifier cannot tell if $p_j = v_j$, it cannot tell between the first and second run, except with negligible probability in which the secrecy of the commitment scheme fails. Suppose the verifier delays it's message beyond β_ℓ with probability p at the first run. It then delays its message with probability at least $p - \varepsilon$ for some negligible fraction ε in the second run. The probability that the simulator does not solve the proof is thus at most $p \cdot (1 - p + \varepsilon)$, which is at most $1/4 + \varepsilon \leq 1/3$, and we are done. ■

If the round has started and ended in the rewind interval, then it it possible for the simulator to perform that rewind. After which it will probably be possible to "solve" that round, and the whole proof with it. (There is one fine point here, however. The verifier commits in the beginning of the protocol. Thus a rewind must be careful only not to rewind a verifier back past the commitment point, which is β_i before the first round. Thus when the simulator rewinds a group of verifiers, it cannot assume, that there is no verifier in the first round of the proof.)

To make the former more formal we need a couple of definitions:

Recall that in the simulation each rewind erases the last part of the transcript. Therefore at any point in the simulation there is a current simulation transcript. The following definitions and lemmas will refer to that transcript in specific points of the simulation.

Definition 7.4 *We denote a rewind $(T \leftarrow T + 2\beta_\ell)$ of level ℓ as* valid *in a partial simulation transcript if for any later rewind $(T' \leftarrow T'')$, $T' \geq T$*

Definition 7.5 *We denote a partial simulation transcript as* solvable *if each of its proofs is rewound at least once in each proof round, except, maybe, a round of level j in which the prover message was sent less than $2\beta_j$ time units before the time of the transcript.*

Definition 7.6 *Denote a partial simulation transcript as ℓ-solvable if each of its proofs of level $\geq \ell$ is rewound at least once in each proof round, in a valid rewind, except, maybe, a round of level j in which the prover message was sent less than $2\beta_j$ time units before the time of the transcript.*

Definition 7.7 *We say that rewind $r' = (T' \leftarrow T' + 2\beta_j)$* invalidates *rewind $r = (T \leftarrow T + 2\beta_\ell)$ if r' comes after r in the simulation, before performing r' r was valid, and after performing r', r is no longer valid.*

Note that at most one rewind can invalidate a given rewind.

From the following lemma it also follows that whenever a proof preamble is complete , it has been rewound on each of its rounds:

Lemma 7.8 *For every partial session transcript R of run σ of protocol Π, R is solvable.*

Proof: We will show by induction that the session transcript is always ℓ-solvable, beginning from $\ell = L$, where L is the maximal level in the simulation:

Base: For $\ell = L$: At the beginning the transcript no proofs have yet been started, and thus it is valid.

In the top level, there are cycles of the following: progress of $2 \cdot \beta_L$ of the simulation, followed by a rewind of $2 \cdot \beta_L$ and then progress of β_L. Therefore there is a rewind every round. from lemma 7.9 it follows that no rewind is invalidated by lower-level rewinds.

The series of level-L rewinds will certainly never invalidate one another, as each one's ending point is exactly β_L after the previous one.

Step: Assume that R is $\ell+1$-solvable. We now need to show that R is ℓ-solvable. We only need to show that there is a valid level-ℓ rewind every round.

Let's look at a rewind $r = (T \leftarrow T + 2\beta_{\ell+1})$ that is valid in the current transcript. According to the rewind schedule it should be followed by a call to **simul**$(T, 2\beta_{\ell+1}, \ell)$. In this run $\frac{length}{\beta_\ell} = \frac{\beta_{\ell+1}}{\beta_\ell} = 2$ level-ℓ rewinds $(T \leftarrow T + 2\beta_\ell)$ are performed.

Those rewinds are valid in the current transcript, because their starting point is $\geq T$ and they come immediately after r, except some lower-level rewinds in between. But those lower-level rewinds cannot invalidate the level-ℓ rewinds, according to lemma 7.9 and thus any rewind that would have invalidated them would have also also invalidated r.

■

Lemma 7.9 *A rewind $r' = (T' \leftarrow T' + 2\beta_j)$ (of level j) can not invalidate any rewind $r = (T \leftarrow T + 2\beta_\ell)$ for any $\ell > j$.*

Proof: For r' to invalidate r, r' must come after r in the simulation and also $T' < T$. We will show that if this is the case, then there has been at least one rewind of level $> \ell$ before r that has invalidated r:

According to the rewind schedule, a rewind always follows a progress of the same amount of the protocol. This means that before r' was performed, the simulation was at T' (and progressed to $T + 2\beta_j$). Therefore rewinds never "accumulate".

After r is performed, **simul**() is called for level $\ell - 1$ for a period of β_ℓ . In this call, and all of its recursive calls, no rewind can be performed back beyond T, because every rewind comes after a simulation of the same period. After the call to **simul**() the simulation is at time $T + \beta_\ell$.

Thus for r' to invalidated r has to begin after the simulation has reached $T + \beta_\ell$ again. Again, because rewinds cannot accumulate due to the advancement before every rewind, r''s starting point cannot be $< T + \beta_\ell$. Thus r''s must be more than $\beta_\ell = 2 \cdot \beta_{\ell-1}$. Therefore r' cannot be of level $< \ell$.

■

References

[1] B. Barak. How to get beyond black-box simulator barrier. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS-2001)*, Las Vegas, Nevada, October 2001.

[2] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *JCSS*, 37(2):156–189, October 1988.

[3] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In ACM, editor, *Proceedings of the thirty second annual ACM Symposium on Theory of Computing (STOC-2000)*, pages 235–244, Portland, Oregon, USA, May 2000. ACM Press.

[4] Ran Canetti, Joe Killian, Erez Petrank, and Alon Rosen. Concurrent zero-knowledge requires $\tilde{\Omega}(\log n)$ rounds. In *Proceedings of the thirty third annual ACM Symposium on Theory of Computing (STOC-2001)*. ACM Press, July 2001.

[5] Giovanni Di Crescenzo and Rafail Ostrovsky. On concurrent zero-knowledge with preprocessing. In Michael Wiener, editor, *Advances in Cryptology – CRYPTO '99*, Lecture Notes in Computer Science, pages 485–502. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1999.

[6] Ivan Damgård, Torben P. Pedersen, and Birgit Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In Douglas R. Stinson, editor, *Proc. CRYPTO 93*, pages 250–265. Springer, 1994. Lecture Notes in Computer Science No. 773.

[7] Ivan Bjerre Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT ' 2000*, Lecture Notes in Computer Science, pages 418–430, Brugge, Belgium, 2000. Springer-Verlag, Berlin Germany.

[8] C. Dwork and A. Sahai. Concurrent zero-knowledge: Reducing the need for timing constraints. *Lecture Notes in Computer Science*, 1462:442–457, 1998.

[9] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *Proceedings of the thirtieth annual ACM Symposium on Theory of Computing (STOC-98)*, pages 409–418, Dallas, Texas, May 1998. ACM Press.

[10] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the twenty-second annual ACM Symposium on Theory of Computing, Baltimore, Maryland, May 14–16, 1990*, pages 416–426, New York, NY, USA, 1990. ACM Press.

[11] Uriel Feige. *Alternative models for zero knowledge interactive proofs.* PhD thesis, Weizmann Institute of science, 1990.

[12] O. Goldreich. Foundation of cryptography — fragments of a book. Available from the *Electronic Colloquium on Computational Complexity (ECCC)* `http://www.eccc.uni-trier.de/eccc/`, February 1995., February 1995.

[13] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 9(3):167–189, Summer 1996.

[14] Oded Goldreich and Hugo Krawczyk. On the composition of Zero-Knowledge Proof systems. *SICOMP*, 25(1):169–192, 1996.

[15] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *ACM Symposium on Theory of Computing (STOC '85)*, pages 291–304, Baltimore, USA, May 1985. ACM Press.

[16] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive systems. *SIAM Journal of Computing*, 18(1):186–208, 1989.

[17] Joe Kilian. Zero-knowledge with log-space verifiers. In *29th annual Symposium on Foundations of Computer Science, October 24–26, 1988, White Plains, New York*, pages 25–35, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1988. IEEE Computer Society Press.

[18] Joe Kilian and Erez Petrank. Concurrent zero-knowledge in poly-logarithmic rounds. In *Proceedings of the thirty third annual ACM Symposium on Theory of Computing*. ACM Press, 2001.

[19] Joe Kilian, Erez Petrank, and C. Rackoff. Lower bounds for zero knowledge on the Internet. In *39th Annual Symposium on Foundations of Computer Science: proceedings (FOCS-98)*, pages 484–492, Palo Alto, California, November 1998. IEEE Computer Society Press.

[20] Alon Rosen Manoj Prabhakaran and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *Proceedings of the 43th Annual Symposium on Foundations of Computer Science*, Vancouver, Canada, November 2002.

[21] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[22] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *21th Annual Symposium on Theory of Computing (STOC)*, pages 33–43. ACM Press, 1988.

[23] Y. Oren. On the cunning powers of cheating verifiers: Some observations about zero knowledge proofs. In Ashok K. Chandra, editor, *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 462–471, Los Angeles, CA, October 1987. IEEE Computer Society Press.

[24] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. *Lecture Notes in Computer Science*, 1592:415–431, 1999.

[25] Alon Rosen. A note on the round-complexity of concurrent zero-knowledge. In *CRYPTO: Proceedings of Crypto*, 2000.

סיבוכיות זמן סיבובים ואפס מידע אסינכרוני

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת תואר

מגיסטר למדעים

מדעי המחשב

צפריר כהן

הוגש לסנט הטכניון — מכון טכנולוגי לישראל

שבט תשס"ג חיפה ינואר 2003

המחקר נעשה בהדרכת ד"ר ארז פטרנק

הפקולטה למדעי המחשב

הכרת תודה

ברצוני להודות למנחה שלי, ארז פטרנק, על העזרה לאורך כל שלבי הכנת העבודה.
כמוכן אני מבקש להודות לאמי על העידוד, ולדן קניגסברג ועידו טל על העזרה הטכנית והעצות.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי

תוכן עניינים

תקציר באנגלית ix

רשימת סמלים xi

1 מבוא **1**

1.1 סיבוכיות זמן-תגובה 1

1.2 אפס מידע מקבילי 3

1.2.1 פרוטוקולים מרובי משתתפים 5

1.3 טכניקות 5

1.4 תרומות 6

1.5 עבודות דומות 6

1.6 מינוח 7

1.7 מדריך לעבודה הזו 7

2 מושגים **9**

2.1 הוכחות באפס מידע 9

2.2 התחייבויות ביטים 10

2.3 אי-הבחנה בין עדים 11

2.4 סימולצית קופסה-שחורה 12

2.5 אפס מידע אסינכרוני 13

2.6 פרמטרי הסיבוכיות 14

3 תוצאה עיקרית **15**

4 סקירת הפרוטוקול וההוכחה **17**

5 הפרוטוקול **19**

6 הסימולטור **23**

6.1 תזמון ההחזרות 24

6.2 ההשפעות של ביצוע החזרות 27

7 ניתוח הסימולטור **29**

7.1 יעילות 29

7.2 אי-הבחנה 30

רשימת מקורות **35**

תקציר ix

רשימת איורים

5.1 הפרוטוקול 20

6.1 תיאור של תזמון ההחזרות בסימולטור 25
6.2 תזמון ההחזרות בריצה של סיבוב אחד מרמה 3 26
6.3 הדגמת החזרה אחת 28

תקציר

בעבודה זו אנו מציגים שני נושאים: ראשית אנו בוחנים מדד סיבוכיות חדש לפרוטוקולים, הקשור לסיבוכיות הסיבובים. מדד זה הוא *סיבוכיות זמן תגובה*. הוא מתייחס לזמן הריצה של הפרוטוקול, וכן לזמן התגובה של כל אחד מהצדדים. לאחר מכן אנו משפרים את תוצאות הסיבוכיות הנוכחיות ביחס למדד זה:[1] אנו משיגים פרוטוקול אשר סיבוכיות זמן התגובה שלו היא כמעט אופטימלית.

סיבוכיות זמן תגובה

נהוג להשתמש במספר הסיבובים של פרוטוקול כמדד סיבוכיות. ההנחה היא שרוב הזמן מתבזבז על תקשורת (זמן החישוב זניח ביחס לזמן התקשורת). הנחה נוספת היא שזמן סיבוב הוא בקירוב קבוע. ההנחה השניה (זמן סיבוב קבוע בקירוב) לא תמיד עומדת במבחן המציאות.

מקרה אחד מעניין שבו היא אינה תקפה, הוא כאשר אחד הצדדים מעכב בכוונה את תשובותיו. כזה הוא, למשל, המוודא בדוגמה הבעייתית שהוצגה [9], שנראה בהמשך.

מדד סיבוכיות זמן-התגובה הוא מדד אשר מתייחס לזמן-תגובה זה של כל אחד מהצדדים: מי שתמיד הגיב יותר מהר, יקבל הבטחה טובה יותר ממי שלפעמים מאחר בתגובותיו. ההגדרה הפורמלית:

הגדרה 7.10 ***זמן תגובה בסיבוב**: עבור משתתפים שונים A, B בסיבוב i של הריצה המסוימת σ של פרוטוקול Π נאמר שזמן התגובה של A הוא t אם לפי הריצה σ של Π A צריך לשלוח הודעה בסיבוב i ו-t הוא משך פרק הזמן המתחיל בשליחת ההודעה בסיבוב ה-$i-1$ ע"י B ומסתיים בקבלת ההודעה בסיבוב ה-i ע"י B. אם A לא אמור לשלוח הודעה בסיבוב i, זמן התגובה שלו בסיבוב זה הוא 0.*

הגדרה 7.11 ***זמן תגובה**: עבור משתתפים שונים A, B בריצה המסוימת σ של פרוטוקול Π נאמר שזמן התגובה של A הוא t אם t הוא המקסימום על זמני התגובה של A בכל הסיבובים של σ.*

נשים לב שזמן התגובה נמדד בשעונו המקומי של B.

[1] אולם ר' גם סעיף "עבודות קשורות`` לגבי עבודות עדכניות

הגדרה 7.12 ***סיבוכיות זמן תגובה**: עבור משתתף A בפרוטוקול Π נאמר שסיבוכיות זמן התגובה של הפרוטוקול ביחס ל-A היא t אם בכל ריצה אפשרית σ של Π סה"כ זמן התקשורת אינו עולה על $t \cdot m$.*

מההגדרות נובע שאם אורך כל סיבוב הוא קבוע, אזי סיבוכיות זמן התגובה שווה לסיבוכיות הסיבובים, ביחס לכל אחד משני הצדדים.

מטרתנו היא לבנות פרוטוקול שבו מוודא אשר מעכב את הודעותיו, נענש על כך[2]. אולם נשים לב שפרוטוקול אשר מבטיח סיבוכיות זמן-תגובה מסוימת יכול למלא תפקיד מועיל גם בסביבה של רשת הטרוגנית או לא יציבה: בסביבה כזו ישנן השהיות רשת שקשה לחזות מראש. משתתפים בעלי חיבור "איטי`` ייענשו אמנם, אולם לכל היותר ביחס ישר להשהיית הרשת שלהם.

נדגים את שימושיות המושג ע"י הצגת פרוטוקול להוכחה באפס מידע אסינכרוני[3]. הפרוטוקול אמנם יהיה בעל סיבוכיות סיבובים של $\tilde{O}\left(\log^2 n\right)$[4], אולם סיבוכיות זמן התגובה תהיה $\tilde{O}\left(\log n\right)$, וכך נראה שזמן הריצה הוא אמנם $\tilde{O}\left(\log n\right)$.

פרוטוקולים מרובי-משתתפים

אם ננסה להרחיב את הגדרת סיבוכיות זמן-תגובה(7.12) לפרוטוקולים מרובי משתמשים נתקל בבעיה: הגדרת זמן-התגובה (7.10) מניחה שהפרוטוקול הוא פרוטוקול "פינג-פונג`` בין שני משתתפים. לכן נצטרך להגדיר מחדש את זמן התגובה.

הגדרה 7.13 ***הודעות קודמות:** לכל הודעה M אשר נשלחת בפרוטוקול ממחשב A אנו מסמנים כהודעות קודמות ל-M את קבוצת כל ההודעות אשר התקבלו ע"י A לפני ש-M נשלחה. אנו מסמני קבוצה זו ב-B_M*

הגדרה 7.14 ***זמן תגובה להודעה:** לכל הודעה M שנשלחה בפרוטוקול ממחשב A אנו מסמנים כ-זמן תגובה את הזמן בין מאורע שליחת ההודעה האחרון של הודעה ב-B_M לבין קבלת M ע"י יעד ההודעה.*

הגדרה 7.15 **זמן תגובה (גרסה מרובת-משתתפים):** *זמן התגובה של מחשב A בריצה σ של פרוטוקול Π הוא המקסימום על זמני התגובה של כל ההודעות ש-A שלח בפרוטוקול.*

בעוד שבהגדרת זמן התגובה לפרוטוקול פשוט (הגדרה 7.10) הזמן נמדד בשעונו המקומי של מחשב אחד, בהגדרות אילו הזמן הוא זמן של מאורעות שמתרחשים אצל כמה מחשבים שונים. לכן ההגדרות מניחות שקיים צופה אשר מסוגל למדוד זמן של כל המאורעות במערכת.

---

[2] ההנחה בפרוטוקול זה היא שהמוכיח הוא בקירוב הגון, ולא צריך להגן על המוודאים ממצב שבו המוכיח מעכב הודעות ללא סיבה

[3] בחרנו לתרגם את המונח האנגלי $concurrent$ zero knowledge ל-"אפס מידע אסינכרוני``. המונח המקורי נשמע קרוב מדי ל-parallel zero knowledge

[4] הכוונה בסימונים $\tilde{O}(\cdot)$ ו-$\tilde{\Omega}(\cdot)$ היא הזנחת איברים כופלים אשר הם לכל היותר בשיעור פולילוגריתמי מהאיבר הראשי. מכאן ש-$\tilde{O}\left(\log^2 n\right)$ הוא כל מה שהוא [לכל היותר] $O\left(\log^2 n \left(\log\log\right)^k n\right)$

אפס-מידע אסינכרוני

הוכחות באפס מידע הן הוכחות אינטראקטיביות אשר בהן לא עובר מידע מהמוכיח (prover) למוודא (verifer) למעט עצם הטענה שמוכחת. את זאת מראים ע"י בניית סימולטור (simulator) פולינומי אשר מייצר תעתיקי הוכחות בהתפלגות בלתי-ניתנת להבחנה מהתפלגות ההוכחות המקוריות. המושג "הוכחות באפס מידע" הוצג לראשונה ע"י גולדוואסר, מיקאלי וראקוף [16].

כמעט כל פרוטוקול הוכחה באפס-מידע הוא מסוג *קופסה שחורה*: בהגדרת אפס-מידע מותר ליצור אלגוריתם סימולטור נפרד עבור כל כל מוודא אפשרי. אולם בפועל כמעט בכל ההוכחות ישנו אלגוריתם מוודא כללי אשר משתמש במכונת המוודא הנתונה כקופסה שחורה: רק מריץ אותה על קלטים כדי לקבל פלטים. כמעט כל החסמים התחתונים מתייחסים רק להוכחה מסוג זה, ואמנם קשה לבנות סימולטור אשר יצליח לנתח את פעולתו של אלגוריתם המוודא שלא ע"י הרצה.

כדי לנצל את המידע שהתקבל מההרצה, הסימולטור משתמש ביתרון שיש לו על המוכיח האמיתי: הסימולטור יכול תמיד להתחיל מחדש את הסימולציה, אם הוא נקלע למצב לא טוב. יתרה-על-כן, הסימולטור יכול לזכור את המצב בנקודות מסוימות, ולהמשיך את ההרצה מהן: הוא הרי שולט על כל פרמטרי ההרצה. במקרה כזה שבו הסימולטור מפסיק את ההרצה הנוכחית וחוזר לנקודה קודמת בסימולציה אנו קוראים *החזרה* (*rewind*).

אולם ההגדרה המקורית מתייחסת למודל תקשורת מוגבל למדי, שבו המוכיח והמוודא מריצים את ההוכחה בלי קשר לשאר העולם (גם במודל ה-auxilary input מ-[14] אין אפשרות להשפעה של הוכחה אחת על הוכחה אחרת, כמו במקרה הבא). סביבת רשת מודרנית היא אסינכרונית מטבעה. לדוגמה, במקרים רבים צפיה ב"דף`` אחד בדפדפן דורשת כמה וכמה התקשרויות לכמה שרתים.

בשנים האחרונות מספר עבודות בדקו את העמידות של תכונת אפס-המידע במודל תקשורת כללי ומציאותי יותר. דוורק, נאור וסהאיי [9] טבעו את המונח *concurrent zero knowledge* (תורגם על-ידינו *לאפס-מידע אסינכרוני, כאמור*) עבור פרוטוקול הוכחה עם תכונת אפס-מידע, אשר שומר על תכונה זו גם אם מרכיבים ביחד מספר פולינומי של הוכחות בכל תזמון אפשרי. כמוכן הם הראו שכמה מהפרוטוקולים המקובלים לאפס-מידע אינם מקיימים אפס-מידע אסינכרוני.

קיליאן, פטרנק ורקוף[19] הראו את החסם התחתון הראשון: כל שפה שיש לה הוכחה באפס מידע (קופסה שחורה) בארבעה סיבובים היא ב-$\mathcal{BPP}$. רוזן [25] שיפר את החסם ל-7 סיבובים, וקנטי, קיליאן, פטרנק ורוזן [4] העלו את החסם ל-$\tilde{\Omega}(\log n)$.

נשים לב שאפשר להשתמש בהוכחה של [4] כדי להראות את אותו החסם עבור סיבוכיות זמן-התגובה של המוודא. זאת מכיוון שההוכחה משתמשת בתזמון שבו אורך סיבוב הוא קבוע.

פרוטוקולים כאלו קיימים: ריצ'רדסון וקיליאן [24] הראו פרוטוקול אפס-מידע אסינכרוני עם מספר פולינומי של סיבובים. קיליאן ופטרנק [18] שיפרו את הניתוח של הפרוטוקול שהוצע ב-[24], והראו שלמעשה מספיקים $\tilde{O}\left(\log^2 n\right)$

סיבובים.

הפרוטוקול

במאמר [9] יש הצעה לפרוטוקול אפס-מידע אסינכרוני. אולם לפרוטוקול זה יש מגבלה אחת חמורה: לאחד הסיבובים ישנו פרק זמן קצוב מראש, t.[5] המוכיח יכול להשתמש בכך כדי לענות למוודאים לפי הסדר שבו הם שלחו את הודעותיהם. למוכיח זה לא משנה, אולם לסימולטור (אשר צריך מדי פעם לבצע החזרות) הדבר מונע היווצרות של סידרה אקספוננציאלית של החזרות.

הפרוטוקול שלנו הוא נסיון להשתמש במגבלות התזמון הללו, אולם לאפשר עדיין לכל מוודא, בעל כל זמן תגובה, לסיים את הפרוטוקול. בפרוטוקול שלנו יש מגבלות תזמון דומות. אולם אם מוודא לא ענה בזמן, הוא לא נפסל לחלוטין: זמן התגובה שלו מוכפל, והוא מתחיל מחדש את הפרוטוקול. המוכיח נדרש לזכור את זמני התגובה של כל המוודאים.

הפרוטוקול הבסיסי שבו אנו משתמשים הוא הפרוטוקול של [24]. זהו פרוטוקול המחולק לשני שלבים: "הקדמה``, ו"חלק עיקרי``. בניגוד למשתמע משמם, ההקדמה היא בעלת מספר ארוך של סיבובים, ואילו מספר הסיבובים בחלק העיקרי הוא קבוע. ההקדמה אינה תלויה בטענה המוכחת מטרתה היא לספק לסימולטור בעל יכולת ביצוע החזרה אפשרות לקבל עד לחלק העיקרי. בהקדמה יש $k = \tilde{O}(\log n)$ סיבובים. בתחילתה המוודא מתחייב על k מחרוזות אקראיות. לאחר מכן, בכל סיבוב של ההקדמה המוכיח מתחייב על ניחוש לערך של אחת המחרוזות, ולאחר מכן המוודא מגלה את המחרוזת. בחלק העיקרי המוכיח מוכיח שאו שהטענה המקורית נכונה, או שאחד הניחושים שלו הצליח.

המוכיח המקורי יכול פשוט להוכיח את הטענה. הוא אפילו לא צריך את ההקדמה. לסימולטור מספיקה היכולת להחזיר לפחות סיבוב אחד של הפרוטוקול מאחרי זמן חשיפת המחרוזת ללפני זמן ההתחייבות: ברגע שידוע לו ערך המחרוזת, הוא מחזיר לאחור את הסימולציה ומתחייב על ערך זה כניחוש.

אולם שלא כמו [18] אנו מטילים מגבלות תזמון (רק על סיבובי ההקדמה): מגדירים מראש השהית סיבוב, T. המוודא חייב לענות לאחר זמן T. המוכיח יחזיר תשובה לאחר פרק זמן של בדיוק $2T$. אם המוודא עונה באיחור, הוא מתחיל מהתחלה, עם השהיה כפולה (ואם גם השהיה כפולה זו לא תספיק, ההשהיה שוב תוכפל בפעם הבאה שהוא יענה מאוחר מדי, וכך הלאה).

כלומר יש כאן אוסף של *רמות* בפרוטוקול: בכל רמה כל המוודאים מקבלים תשובה מהמוכיח בהשהיה קבועה. לכן בכל רמה יש סדר כמו בפרוטוקול של [9]. כמוכן מספר הרמות הוא לכל היותר לוגריתמי: לאחר מספר סופר-לוגריתמי של הכפלות, השהיית סיבוב היא כבר סופר-פולינומיאלית.

[5] המגבלות קצת שונות, אולם הן פושטו לצורך הצגת הבעיה

עבודות קשורות

העבודה שלנו על סיבוכיות זמן-תגובה קשורה, כמובן, לספרות העשירה העוסקת באלגוריתמים מבוזרים, וממשיכה את המחקר של אפס-מידע בסביבה מבוזרת.[6] אולם נשים לב שבעוד שבמודל המבוזר הרגיל התוקף יכול להסתפק בהצלחה בחלק מהמקרים, אנו דורשים שכל הצדדים ייתנו פלט "נכון`` בסוף הפרוטוקול. במודל שלנו אנו מוסיפים דרישות על מועד הסיום של המשתתפים השונים: משתתף זריז יכול לסיים את הפרוטוקול זמן רב לפני השאר (בהקשר זה "פרוטוקול`` הנו אוסף כל ההוכחות האינטראקטיביות).

ישנם כמה כיוונים של נסיונות לעקוף את המגבלות של החסם התחתון. מספר עבודות הטילו מגבלות על האסינכרוניות של המערכת ([9, 8, 5]). קבוצה אחרת מניחה הנחות על סביבת הפעולה, לדוגמה: מערכת מפתח ציבורי ([7, 3])

אחת העבודות המעניינות שפורסמו לאחרונה היא זו של ברק ([1]). בעבודה זו הוא מראה מערכת הוכחה באפס מידע (אסינכרוני) שבה הסימולטור אינו מסוג קופסה שחורה, ולכן החסם לא תקף לגביו. ואמנם מתקבל פרוטוקול עם מספר קבוע של סיבובים. סימולטור זה אכן נדרש "לנתח`` במידה מסוימת את דרך פעולת המוודא.

אולם הבעיה העיקרית בשימוש בפרוטוקול של עבודה זו היא שמספר המוודאים עבורם הפרוטוקול יוכל להיות עמיד הוא פרמטר של הפרוטוקול שצריך להיות ידוע בתחילת הריצה. יתר-על-כן גודל ההודעות בפרוטוקול תלוי (לינארית) בפרמטר זה.

במאמר שיצא לאחר פרסום עבודה זו, הצליחו פרבהאקראן, רוזן וסהאיי [20] להראות פרוטוקול שעובד ב-$\tilde{O}(\log n)$ סיבובים במודל הרגיל. במאמרם הם השתמשו בניתוח אחר לסימולטור של [18] (אם כי תוך שימוש בפרוטוקול שונה במקצת). בגלל שהפרוטוקול קצת שונה הם יכלו לשנות במקצת את ההנחות שעליה הם הסתמכו, אולם גם סיבוכיות התקשורת של הפרוטוקול גדלה.

[6] ואכן לא נופתע אם כבר הוצעו הגדרות דומות בספרות

סיבוכיות זמן סיבובים ואפס מידע אסינכרוני

צפריר כהן