Incrementally Verifiable Computation
or
Knowledge Implies Time/Space Efficiency

by

Paul Valiant

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2007

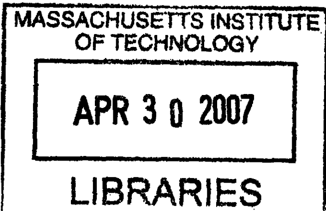
© Paul Valiant, MMVII. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author ....
Department of Electrical Engineering and Computer Science
February 2, 2007

Certified by ....
Silvio Micali
Professor of Computer Science
Thesis Supervisor

Accepted by ....
Arthur Smith
Chairman, EECS Committee on Graduate Students


MASSACHUSETTS INSTITUTE OF TECHNOLOGY
APR 3 0 2007
LIBRARIES

ARCHIVES

Incrementally Verifiable Computation

or

Knowledge Implies Time/Space Efficiency

by

Paul Valiant

Submitted to the Department of Electrical Engineering and Computer Science
on February 2, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

The probabilistically checkable proof (PCP) system enables proofs to be verified in time polylogarithmic in the length of a classical proof. Computationally sound (CS) proofs improve upon PCPs by additionally shortening the length of the transmitted proof to be polylogarithmic in the length of the classical proof. In this thesis we explore the ultimate limits of non-interactive proof systems with respect to time/space efficiency and the new criterion of composability.

We deduce the existence of our proposed proof system by way of a natural new assumption about proofs of knowledge. In fact, a main contribution of our result is showing that knowledge can be "traded" for time and space efficiency in noninteractive proof systems.

Thesis Supervisor: Silvio Micali
Title: Professor of Computer Science

Acknowledgments

I would like to thank Madhu Sudan, Brendan Juba and Rafael Pass for many helpful discussions during the course of this work. I would especially like to thank Silvio Micali for his tireless help and support in all stages of this project, from the initial problem formulation to the final editing.

Paul Valiant

February, 2007

Contents

1 Introduction **9**

1.1 A new problem . . . 9

1.2 Intuitive idea of our solution . . . 10

1.3 Proofs of knowledge . . . 11

2 Definitions **15**

2.1 Noninteractive proofs and the Common Random String model . . . 15

2.2 The notion of incrementally verifiable computation . . . 16

2.3 Noninteractive CS proofs of knowledge . . . 18

3 CS proofs of knowledge in the random oracle model **21**

3.1 Witness-extractable PCPs . . . 21

3.2 CS proof construction . . . 23

4 Proofs in the Common Reference String model **29**

4.1 Merging proofs . . . 29

4.2 Incrementally verifying computation . . . 31

Chapter 1

Introduction

Perhaps the simplest way to introduce the computational problem we address is by means of the following.

Human motivation. Suppose humanity needs to conduct a very long computation which will span super-polynomially many generations. Each generation runs the computation until their deaths when they pass on the computational configuration to the next generation. This computation is so important that they also pass on a proof that the current configuration is correct, for fear that the following generations, without such a guarantee, might abandon the project. Can this be done?

Computational setting In a more computational context, this problem becomes: How can we compile a machine M into a new machine M' that frequently outputs pairs (c_i, π_i) where the ith output consists of the ith memory state c_i of machine M, and a proof π_i of its correctness, while keeping the resources of M intact?

1.1 A new problem

We motivate our problem by way of a few examples of how current techniques fail to achieve our goal. Suppose we are given a computation M that takes time t and space k.

The natural thing to do is have the compiled machine M' keep a complete record of all the memory states of M it has simulated so far; every time it simulates a new state of M, it uses this record to output a proof that its simulation of M is thus far correct. However, this approach has the clear drawbacks that the compiled machine M' uses space tk to store the records, and the proofs it outputs consist simply of this record of size tk; this requires the verifier of the proofs to also use time tk and space tk to verify *each* proof. If t is polynomial in k, then all these parameters are polynomial in k and the system is not so bad; however, we concern ourselves here with the case where the running time t is typically much larger than k, in which case this naive system is not at all efficient. What we need is a more efficient proof system.

We note that the problems of improving the efficiency of the construction, transmission, and verification of proofs have been important themes in our field, and have fueled a long line of research. One major milestone on this path was the discovery of *probabilistically checkable proofs* (PCPs) (see [1, 2, 4, 9] and the references therein). Under the PCP proof system statements with classical proofs of exponential length could now be verified in polynomial time, via randomized sampling of an encoded version of the classical proof. The PCP system still uses exponential resources to construct and transmit the proof, but verification is now polynomial time.

The second milestone we note is the theory of *computationally sound* (CS) proofs as formalized by Kilian and Micali [11, 12]. This proof system improves on the PCP system by keeping verification polynomial time while shortening the transmitted proof from exponential to polynomial length. If we instruct the compiled machine M' to output CS proofs, then the length of the transmitted proofs, and the time and space required by the verifier are now polynomial in k, but the compiler still requires memory at least t, and a time interval of at least t between consecutive proofs.

1.2 Intuitive idea of our solution

The ideal way to achieve incrementally verifiable computation consists of *efficiently merging two CS proofs of equal length into a single CS proof* which is as long as and

as easy to verify as each of the original ones. Letting $c_0, c_1, \ldots$ be the sequence of configurations of machine M, and for $i < j$, intuitively denote by $c_i \stackrel{cs}{\rightarrow} c_j$ a CS proof that configuration c_j is correctly obtained from configuration c_i by running M for $(j - i)$ steps. After running M for 1 step from the initial configuration c_0 so as to reach configuration c_1 one could easily produce a CS proof $c_0 \stackrel{cs}{\rightarrow} c_1$. Running M for another step from configuration c_1, one can easily produce a CS proof $c_1 \stackrel{cs}{\rightarrow} c_2$. At this point, if CS proofs can be easily merged as hypothesized above, one could obtain a CS proof $c_0 \stackrel{cs}{\rightarrow} c_2$. And so on, until a final configuration c_f is obtained, together with a CS proof $c_0 \stackrel{cs}{\rightarrow} c_f$.

Unfortunately, we have no idea of how to achieve such efficient and length preserving merging of CS proofs. However, if a variant of CS proofs – which we call CS *proofs of knowledge* – exist, we show a sufficient approximation of this ideal strategy.

A Dynamic CS-Proof Data Structure. In essence, we show that while machine M marches through configurations $c_1, c_2, \ldots$, it can carry along a set S of CS proofs of the form $c_i \stackrel{cs}{\rightarrow} c_j$ such that:

- Each element of S has length $poly(k)$ whenever M works in space k.
- The cardinality of S always is poly-logarithmic in t, M's running time.
- The final set S constitutes a (novel type of) CS proof that c_f is the final configuration of M, verifiable in time polynomial in k and $\log t$.

In essence, therefore, while we cannot merge any two proofs, we can merge subsets of S sufficiently often.

1.3 Proofs of knowledge

Central to the constructions in this paper is the notion of a *proof of knowledge* [10]. To motivate this notion, consider the following situation: a job applicant is being interviewed for a specific position, and wishes to convince the interviewer that he possesses knowledge relevant to this position. One option he has is to simply list

everything he knows about the area and if the interviewer has the time and patience, he will certainly be convinced of the applicant's knowledge. However, if time is limited, the applicant must try to prove to the interviewer that he possesses knowledge of the right nature without actually divulging it all. Alternatively if time is not a constraint, security may be a constraint: perhaps the applicant is applying for a job predicting the stock market; if he reveals all his knowledge about the future of the stock market before he is hired, the interviewer may reject him and strike it rich on his own.

To formalize the notion of "knowledge of the right nature", we consider a polynomial time relation $R(x, y)$, and for a given x let this knowledge consist of any y such that $R(x, y) = 1$. We note that a proof of knowledge in this sense is also a proof in the classical sense of the statement "$\exists y : R(x, y) = 1$".

To show that the prover contains "knowledge", despite the fact that in his normal operation he may never reveal such knowledge, it is standard to introduce a "knowledge extractor," E, which produces this knowledge at the end of its interaction with the prover. While anyone interacting normally with the prover may never be able to access this knowledge, we consider E principally as a thought experiment, and give it some extra powers beyond those of the verifier so that E may recover the knowledge, and thereby justify to us that the prover contained this knowledge in the first place. In different contexts it may be appropriate to endow E with different powers, including additional computation time or non-black box access to the prover.

Proofs of knowledge may be seen as a restricted form of classical proofs. While classically, proofs of a statement "$\exists y : R(x, y) = 1$" can take a wide variety of non-constructive forms, the proof of knowledge form asserts that the prover knows a valid y. This property will be essential to us later as we show how to merge CS proofs of knowledge: while unrestricted proofs may be non-constructive enough that a pair of them may be impossible to corroborate, proofs of knowledge leave enough of a "paper trail" so that we can reassemble pieces of proofs of related theorems to a new whole.

The Non-Interactive CS Knowledge Assumption In [12], Micali explicitly constructs non-interactive CS proofs given a *random oracle.* In Section 3 we show that a variant on Micali's construction is a proof of knowledge as well. Intuitively, there exists an efficient extractor E that, given a statement X, a CS proof π, and access to the CS prover that produced π, quickly outputs a (classical) proof Π of X. Micali, also defines and uses non-interactive CS proofs in the *common random string* (rather than random-oracle) model. But their existence requires an ad hoc assumption. Our assumption is stronger: namely, we posit the existence of non-interactive CS proofs, in the random string model, that also are proofs of knowledge.

In essence, our assumption states that, in a *specific* construction of non-interactive CS proofs (given in the appendix), it is possible to replace the random oracle with a random string and still preserve the proof of knowledge property of CS proofs. (That is, we do not invoke the random-oracle hypothesis in its general form. As shown by Canetti et al. [7] and others in different contexts, we expect that there may be other non-interactive CS proof constructions for which no way to replace the oracle exists.)

Knowledge $\Rightarrow$ time/space efficiency In this work we start with an unusual and very strong assumption about (proofs of) knowledge and conclude with a proof system of unprecedented time and space efficiency. Here we wish to draw the reader's attention not to the assumption or the conclusion, but to the nature of the relationship between them. On the left we make an assumption about *knowledge* in CS proofs: we take a restricted system that only deals with witnesses of length $3k$ and compresses them to proofs of length k, the security parameter, and assume that there is a linear-time *knowledge extractor* that can extract the witness given access to the prover. On the right we conclude with a proof system that compresses any proof to length $poly(k)$, uses space polynomial in the space needed to classically accept the language, and is time-efficient in the tightest possible sense, using $poly(k)$ time to process a step of the classical acceptance algorithm. We note that current constructions of CS proofs based on random oracles need time *polynomial* in the time to classically accept, and space of the same order as their *time*[12]. This constitutes a new technique to leverage *knowledge* to gain time and space efficiency.

Chapter 2

Definitions

2.1 Noninteractive proofs and the Common Random String model

It is a well-known aphorism in cryptography that "security requires randomness". In a standard setting, a participant in a protocol injects randomness into his responses to protect him from some pre-prepared deviousness on the part of the other participant.

In the noninteractive proof setting such an approach is inadequate: the verifier is unable to protect himself with randomized messages to the prover, since he cannot even *communicate* with the prover. To address these issues, the *common random string* (CRS) model was introduced [6, 5].

The CRS model – sometimes called the common *reference* string model – assumes that all parties have access to a random string, and further that each can be confident that this string is truly random and not under the influence of the other parties. Potential examples of such a string are measurements of cosmic background radiation or, for a string that will appear in the future, tomorrow's weather. Typically, protocols in the CRS model use only the randomness present in the common string, and are deterministic given this string.

In the analysis of the security of a CRS protocol leeway must be given for "unlucky" choices of strings, since if every choice of string worked in the protocol we

would not need a random one. Thus even if a CRS protocol has a chance of failing, we still consider it secure if this chance is negligible in the size of the random string.

2.2 The notion of incrementally verifiable computation

Basic notation We denote a Turing machine T with no inputs by $T()$, a Turing machine with one input by $T(\cdot)$, a Turing machine with two inputs by $T(\cdot, \cdot)$, etc. We assume a standard encoding, and denote by $|T|$ the length of the description of T. For a Turing machine T running on input i, we denote by $time_T(i)$ the time T takes on input i, and by $space_T(i)$ the space T takes on input i; we denote the empty input by ϵ, so that $space_T(\epsilon)$ is the space of Turing machine T when run on no input.

The outline We formally define incrementally verifiable computation here. We consider a Turing machine $M()$ that we wish to simulate for t time steps using k memory. We consider a fixed *compiler* $C(\cdot, \cdot)$ that produces from (k, M) an *incrementally verifiable* version of M, namely a machine $C(M, k) = T(\cdot)$ that takes as input the common random string, and runs in time $t \cdot k^{O(1)}$, uses memory $k^{O(1)}$, and every $k^{O(1)}$ time steps outputs its memory configuration m_j and a "proof". The memory configuration output should be interpreted as a claim about the memory configuration of M at the corresponding time. There is a fixed machine V, the verifier, that will accept all pairs of configurations and proofs generated in this way, and will reject other pairs, subject to the usual condition of the CRS model that the verifier may be fooled with negligible probability, and the computational soundness caveat that an adversary with unbounded resources may also fool the verifier.

Incremental timelines and outputs Commonly, Turing machines make an output only once, and making this output ends the computation. Instead, we interpret Turing machines as being able to output their current memory state at certain times in their operation: explicitly, consider a Turing machine with a special state "Out-

put" where whenever the machine is in state "Output" the entire contents of its tape are outputted.

Definition 2.2.1. *An increasing sequence of integers* $\{t_j\}$ *is an* incremental timeline *if there exists an* α *such that for any* j, $t_j - t_{j-1} \leq \alpha$.

If $\{t_j\}$ is an incremental timeline for a specific α, we may refer to $\{t_j\}$ as an *α-incremental timeline.*

Definition 2.2.2. *A Turing machine that makes outputs at every time on an (α-) incremental timeline is called an* (α-) incremental output Turing machine.

Definition 2.2.3 (Feasible Compiler). *Let* $C(\cdot,\cdot)$ *be a polynomial time Turing machine. We say that* C *is a* feasible compiler *if there exists a constant* c *such that for all* $k > 0$ *and all* $M()$ *such that* $|M| < k$, $C(k, M)$ *is a Turing machine* $T(\cdot)$ *with one input satisfying*

1. *T is a k^c-incremental output Turing machine.*
2. *$space_T(r) = k^c$ for all inputs r.*

In other words, properties 1 and 2 guarantee that each compiled machine T outputs its internal configuration "efficiently often" while working in "efficent space."

Definition 2.2.4 (Incrementally Verifiable Computation). *Let* C *be a* feasible compiler, *and let* V*, the verifier, be a polynomial time Turing machine with 5 inputs. The pair* (C, V) *is an* incrementally verifiable computation scheme *secure for computations of length up to* t_k *provided that for every machine* M *as above, the properties below are satisfied.*

Throughout, r *is the common (random) string of length* k^2*. Let the* j*th output of the compiled machine* $C(M, k)$ *be parsed as an ordered pair* (m_j, π_j^r)*, representing a claim about the* j*th memory configuration of* M*, and its proof. We require:*

1. *(Correctness) The compiled machine accurately simulates M, in that m_j is indeed the jth memory configuration of $M(\epsilon)$ for all j, and is independent of r.*

2. *(Completeness) The verifier V accepts the proofs π_j^r: $\forall r, V(M, j, m_j, \pi_j^r, r) = 1$.*

3. *(Computational soundness) For any constant c and for any machine P' that for any length k^2 input r outputs a triple $(j, m_j'^r, \pi_j'^r)$ in time t_k, we have for large enough k that*

$$Prob_r[m_j'^r \neq m_j \wedge V(M, j, m_j'^r, \pi_j'^r, r) = 1] < k^{-c}.$$

2.3 Noninteractive CS proofs of knowledge

We now specify the assumption we make: the existence of noninteractive CS proofs of knowledge.

We note that proofs of knowledge are typically studied in the form of *zero knowledge proofs of knowledge.* In this setting, one party wants to convince another party that he possesses certain knowledge without revealing this knowledge. The reason why he does not simply transmit all his evidence to the other party is that he wishes to maintain his privacy.

In our setting the reason one generation does not just transmit all its evidence to the next generation is not a privacy concern, but rather the concern that the following generation will not have the time to listen to all this evidence.

In both settings, the "knowledge" that must be proven may be considered to be a witness for a member of an NP-complete language: one party proves to the other that he knows, for example, a three-coloring of a certain graph.

In the zero-knowledge setting, our prover does not wish for the verifier to learn a three-coloring of the graph. In the incremental computation setting, our prover is worried that the verifier may not want to spare the resources to learn a three-coloring of the graph.

Related issues were considered in a paper of Barak and Goldreich where they investigated efficient (interactive) ways of providing proofs and proofs of knowledge [3]. Our definition of a noninteractive CS proof of knowledge contains elements from their definition of a *universal argument.*

For the sake of concreteness, we consider a specific NP-complete language, which we define below. This language has the property that for any k the strings in the language of length $4k$ have witnesses of length $3k$. We require of our CS proof system that instead of returning proofs of length $3k$ the proofs are shortened to length k.

Definition 2.3.1 (Noninteractive CS proof of knowledge). *Consider the NP-complete language L_c that consists of the ordered pairs (M, x) where M is a Turing machine and x is a $\{0, 1\}$ string that satisfy the following properties for some k:*

1. $|M| = k$ *and* $|x| = 3k$.
2. *There exists a string w of length $3k$ such that M when run on the concatenation (x, w) accepts within time k^c.*

A noninteractive CS proof of knowledge *is defined by Turing machines P the prover, and U the verifier, a function $K'(k) : \mathbb{Z}^+ \to \mathbb{Z}^+$ that describes the strength of an adversary necessary to break the system for inputs of length k, and constants c, c_1, c_2.*

The tuple (P, U, K', c, c_1, c_2) is a noninteractive CS proof of knowledge if for all machines M of size k and strings x of length $3k$ the following properties hold:

1. *(Efficient prover) For any string r of length k, $time_P(M, x, w, r) = k^{O(1)}$*
2. *(Length shrinking) For any string r of length k, $|P(M, x, w, r)| = k$.*
3. *(Efficient verification) For any string r of length k, $time_U(P(M, x, w, r), M, x, r) \leq k^{c-1}$*
4. *(Completeness) For any string r of length k, $U(P(M, x, w, r), M, x, r) = 1$*
5. *(Knowledge extraction) There is a randomized Turing machine E, the* extractor, *and constant c_2 such that for any machine P' and input (M, x) of length $4k$ such that for all r of length k $time_{P'}(M, x, r) \leq K(k)$ and $Pr_r[U(P'(M, x, r), M, x, r) = 1] = \alpha > 1/K$ we have*

$$Prob[w \leftarrow E(P', M, x) : M(x, w) = 1] > 1/2$$

and the running time of $E(P', M, x)$ is at most $\cdot k^{c_2}/\alpha$ times the expected running time (over choices of r) of $P'(M, x, r)$.

Chapter 3

CS proofs of knowledge in the random oracle model

To introduce CS proofs of knowledge, and support our hypothesis that there exist noninteractive CS proofs of knowledge in the common reference string model we provide details of such proofs in the random oracle model. Specifically, our construction will satisfy Definition 2.3.1 modified by replacing the string r everywhere with access to an oracle R.

The construction of the proofs is based closely on the constructions of Kilian and Micali[11, 12]. The construction of the witness extractor is inspired by the constructions of Pass[13].

3.1 Witness-extractable PCPs

One of the principal tools of the CS proof construction is the probabilistically checkable proof (PCP)[1, 2]. The PCP theorem states that any witness w for a string x in a language in NP can be encoded into a probabilistically checkable witness, specifically, a witness of length n can be encoded into a PCP of length $n \cdot (\log n)^{O(1)}$ with an induced probabilistic scheme (based on x) for testing $O(1)$ bits of the encoding such that:

- For any proof generated from a valid witness the test succeeds.

- For any x for which no witness exists the test fails with probability at least $\frac{2}{3}$.

In practice, the test is run repeatedly to reduce the error probability from $\frac{1}{3}$ to something negligible in n. In addition to the above properties of PCPs, we require one additional property that is part of the folklore of PCPs but rarely appears explicitly:

- Given an appropriate PCP test and a constant γ, from any string s on which the PCP test succeeds with probability at least $1 - \gamma$ we can extract an NP witness w for x.

We sketch briefly how this additional property can be attained. Consider the related notion of a *PCP of proximity* (PCPP)[4]:

Definition 3.1.1 (Probabilistically checkable proof of proximity). *A pair of machines (P, V) are a PCPP for the NP relation $R = \{(x, w)\}$ with* proximity parameter ϵ *if*

- *If $(x, w) \in L$ then the verifier accepts the proof output by the prover:*

$$Prob[V(P(x, w), x) = 1] = 1.$$

- *If for some x, π is ϵ-far from any w such that $(x, w) \in L$, then the verifier will reject π with high probability:*

$$Prob[V(\pi, x) = 1] < \frac{1}{3}.$$

We note that this property is stronger than the standard PCP property since in addition to rejecting if no witness exists, the verifier also rejects if the prover tries to significantly deceive him about the witness. Ben-Sasson et al. showed the existence of PCPPs with $O(1)$ queries and $n \cdot (\log n)^{O(1)}$ length[4]. We use these PCPPs to construct witness-extractable PCPs:

Construction 3.1.2. *Let E be an error-correcting code of constant rate that can correct ϵ fraction of errors, with ϵ the PCPP parameter as above. Let $L = \{(x, w)\}$ be the NP relation for which we wish to find a witness-extractable PCP. Modify L using the code E to a relation*

$$L' = \{(x, E(w)) : (x, w) \in L\}.$$

Let P be a PCPP prover for this relation, which outputs the pair

$$(E(w), P(x, E(w))).$$

The verifier for this proof system is just the PCPP verifier for L', which expects inputs of the form $(E(w), P(x, E(w)))$. Let the witness extractor *for the proof system run the decoding algorithm on the portion of its input corresponding to $E(w)$ and report the result.*

Claim 3.1.3. *Construction 3.1.2 is a witness-extractable PCP with quasilinear expansion, where the verifier reads only a constant number of bits from the proof.*

We note that since E is a constant-rate code and P expands input lengths quasilinearly, this scheme also has quasilinear expansion. Since the PCPP system reads only $O(1)$ bits of the proof, this new system does too.

For any pair $(x, w) \in L$ the proof we generate will be accepted by the verifier, so this scheme satisfies the first property of PCPs. If x is such that no valid w exists for the L relation, then no valid $E(w)$ exists under the L' relation and the verifier will fail with probability at least $\frac{2}{3}$, as required by the second property of PCPs.

Finally, to show the witness extractability property we note that by definition of a PCPP, if the verifier succeeds with probability greater than $\frac{1}{3}$ on the string (s, π) then s is within relative distance ϵ from the encoding of a valid witness $E(w)$. Since the code E can correct ϵ fraction errors, we apply the decoding algorithm to s to recover a fully correct witness w. We have thus constructed a witness-extractable PCP for $\gamma = \frac{2}{3}$.

3.2 CS proof construction

We now outline the construction of noninteractive CS proofs of knowledge, which is essentially the CS proof construction of Kilian and Micali[11, 12]. We present the knowledge extraction construction in the next section.

The main idea of this CS proof construction is for the prover to construct a (witness-extractable) PCP, choose random queries, simulate the verifier on this PCP

and queries, and send only the results of these queries to the real verifier, along with convincing evidence that the queries were chosen randomly and independent of the chosen PCP. For security parameter k' (to differentiate from k used in the definitions of the previous chapeter) the prover sends only data related to k' runs of the PCP verifier, and thus the length of the proof essentially depends only on the security parameter k'.

The technical challenge in the construction is to convince the verifier that the queries to the PCP are independent of the PCP. To accomplish this we use a *random oracle*. Let $\mathfrak{R}$ denote the set of functions

$$R : \{0,1\}^{2k'} \to \{0,1\}^{k'}.$$

By a *random oracle* we mean a function R drawn uniformly at random from the set $\mathfrak{R}$. The machines in our construction will have oracle access to such an R.

We start by defining a *Merkle hash*:

Definition 3.2.1 (Merkle hash). *Given a string s and a function $R : \{0,1\}^{2k'} \to \{0,1\}^{k'}$, do the following:*

- *Partition s into chunks of length k'.*
- *Let each chunk be a leaf of a full binary tree of minimum depth.*
- *Filling up from the leaves, for each pair of siblings s_0, s_1, assign to their parent the string $R(s_0, s_1)$.*

To aid in the notation we define a *verification path* in a tree:

Definition 3.2.2 (Verification path). *For any leaf in a full binary tree, its* verification path *consists of all the nodes on the path from this node to the root, along with each node's sibling.*

The construction of CS proofs is as follows:

Construction 3.2.3. *Given a security parameter k', a polynomial-time relation $L = \{(x, w)\}$ with $|w| < 2^{k'}$ and a corresponding witness-extractable PCP with prover and verifier PP, PV respectively, we construct a CS prover P and verifier V.*

P on input (x, w) and a function $R : \{0,1\}^{2k'} \to \{0,1\}^{k'}$ does the following:

1. *Run the PCP prover to produce $s = PP(x, w)$.*
2. *Compute the Merkle hash tree of s, with s_r the root.*
3. *Using R and s_r as a seed, compute enough random bits to run the PCP verifier PV k' times.*
4. *Run PV k' times with these random strings; let the CS proof $P^R(x, w)$ consist of the $k' \cdot O(1)$ leaves accessed here, along with their complete verification pathways.*

V on input x, a purported proof π and a function R does the following:

1. *Check for consistency of the verification pathways, i.e. for each pair of claimed children (s_0, s_1) verify that $R(s_0, s_1)$ equals the claimed parent.*
2. *From the claimed root s_r run the procedure in steps 3 and 4 of the construction of P, failing if the procedure asks for a leaf from the tree that does not have a verification pathway.*
3. *Accept if both steps succeed, otherwise reject.*

These are essentially the CS proofs of Killian and Micali. In the next section we exhibit the knowledge extraction property of these proofs, and thereby infer their soundness; further properties and applications may be found in the original papers.

Knowledge extraction We now turn to new part of this construction, the *knowledge extractor.*

Recall that we want to construct a machine E that when given a (possibly deceptive) prover P' will efficiently extract a witness w for any x on which

$$Pr[V^R(x, P'^R(x)) = 1] > 1/K.$$

In other words, if P' reliably constructs a proof for a given x, then there is a witness "hidden" inside P', and E can extract one. The general idea of our construction is to simulate $P'^R(x)$ while noting each oracle call and response, construct all possible

Merkle trees that P' could have "in mind", figure out based on the output of P' which Merkle tree it finally chose, read off the PCP at the leaves of the tree, and use the PCP's witness extraction property to reveal a witness.

We note that this extractor is slightly unusual in that it does not "rewind" the computation at any stage, but merely examines the oracle calls P' makes; such extractors have been recently brought to light in other contexts under the names *straight-line extractors*[13] or *online extractors*[8]. The principal reason we need such an extractor is that we require the extractor to run in time *linear* in the time of P', up to multiplicative constant k^{c_2}, and we cannot afford the time needed to match up data from multiple runs.

We show that the following extractor fails with negligible probability on the set of R where $P'^R(x)$ is accepted by the verifier; to obtain an extractor that never fails, we re-run the extractor until it succeeds.

Construction 3.2.4 (CS extractor). *Simulate $P'^R(x)$, and let $q_1, ..., q_t$ be the queries the machine makes to R, in the order in which they are made, duplicates omitted. Assemble $\{q_i\}$ and separately $\{R(q_i)\}$ into data structures that can be queried in time logarithmic in their sizes, $\log t$ in this case. If for some $i \neq j$ $R(q_i) = R(q_j)$, or if for some $i < j$ $q_i = R(q_j)$, then abort.*

Consider $\{q_i\}$ as the nodes of a graph, initially with no edges. For any q_i whose first k' bits equal some $R(q_j)$ and whose second k' bits equal some $R(q_l)$, draw the directed edges from q_i to both q_j and q_l.

In the proof output by $P'^R(x)$ find the string at the root, s_r. If s_r does not equal $R(q_r)$ for some r, then abort. If the verification paths from the proof are not embedded in the tree rooted at q_r, abort.

Compute from x the depth of the Merkle tree one would obtain from a PCP derived from a witness for x. (Recall that in the main text we insist that witnesses have length a fixed fraction of the length of x; in general we could pad witnesses to a prescribed length.) Read off from the tree rooted at q_r all strings of this depth from the root; where strings are missing fill in $0^{2k'}$ instead. Denote this string by pcp.

Apply the PCP witness extractor to pcp, and output the result.

Claim 3.2.5. *Construction 3.2.4 when given (P', x) such that $P'^R(x)$ always runs in time at most $2^{k'/4}$ and that convinces the verifier with probability $Pr_R[V^R(x, P'^R(x)) = 1] = \alpha > 2^{-k'/8}$, will return a witness w for x on all but a negligible fraction of those R on which P' convinces the verifier in time $O(k/\alpha)$ times the expected running time of P'.*

Proof. We show that this construction fails with negligible probability. We begin by showing that the probability of aborting is negligible.

Suppose P' has already made $i - 1$ queries to the oracle, and is just about to query $R(q_i)$. This value is uniformly random and independent of the view of P' at this point, so thus the probability that $R(q_i)$ equals any of q_j or $R(q_j)$ for $j < i$ is at most $2i \cdot 2^{-k'}$. The probability that this occurs for any $i \leq t$ is thus at most $t^2 2^{-k'}$, which bounds the probability that the extractor aborts in the first half of the extractor.

We note that since no two q_i's hash to the same value the trees will be constructed without collisions, and since $i < j$ $q_i \neq R(q_j)$ the graph will be acyclic and thus a valid binary tree. We now must bound the probability that some node on a verification path (including possibly the root) does not lie in the graph we have constructed. Let s_0, s_1 be a pair of siblings on a verification pathway for which the concatenation (s_0, s_1) is not in the graph. Thus P' does not ever query $R(s_0, s_1)$. Since the proof P' generates is accepted by the verifier, the value of $R(s_0, s_1)$ must be on the verification path output by P'. Thus P' must have *guessed* this value without evaluating it, and further, the guess must have been right. This occurs with probability at most $2^{-k'}$. Thus the total probability of aborting is at most $(t^2 + 1)2^{-k'}$.

We now show that if the extractor does not abort, it extracts a valid witness on all but a negligible fraction of R's. Recall that the CS verifier makes k' calls to the PCP verifier, each of which, if seeded randomly, fails with probability $\frac{2}{3}$ whenever the string *pcp* does not encode a valid witness w.

Consider for some non-aborting R and some $i \leq t$ the distribution ρ on R obtained by fixing those values of R that $P'^R(x)$ learns in its first i oracle calls, and letting the values of R on the remaining inputs be distributed independently at random.

Consider an R drawn from the distribution ρ. Construct a Merkle tree from the values $\{(q_j, R(q_j)) : j \leq i\}$ rooted at q_i, i.e., pretending that P', when it finishes, will output $R(q_i)$ as the root, and let *pcp* be the string read off from the leaves, as in the construction of the extractor. Compute from R and $R(q_i)$ as in step 3 of the construction of the CS prover P the k' sets of queries to the PCP verifier. Unless the oracle calls generated here collide with the i previous calls, the PCP queries will be independent and uniformly generated; if witness extraction fails on *pcp* then by definition, these PCP tests will succeed with probability at most $\frac{1}{3}^{k'}$. Adding in the at most $t^2 2^{-k'}$ chance that, under this distribution, one of the new oracle calls will collide with one of the old calls, the total probability that *pcp* is not witness-extractable, yet the tests succeed, is at most $(t^2+1)2^{-k'}$.

Consider all distributions ρ with i fixed values as above. We note that the distributions have disjoint support, since no fixed R could give rise to two different initial sequences of oracle calls. We note also that any R either aborts or induces such a distribution ρ with i fixed values. We now vary i from 1 to t. Consider the set of non-aborting R for which there is some i such that the string pcp_i^R is not witness-extractable yet the PCP tests generated by R all succeed. By the above arguments and the union bound this set has density at most

$$t(t^2+1)2^{-k'}.$$

By assumption the set of R for which the verifier accepts $P'^R(x)$ has density at least $2^{-k'/8}$. Thus for all but a negligible fraction of these R, the string *pcp* is witness-extractable, and we may recover a witness w as desired. □

We note that our extractor runs logarithmic factor slower than P'. Since the running time of P' is subexponential in k, the extractor takes time k factor more than P'. As noted above, if P' returns an acceptable proof with probability α we may have to run the extractor $1/\alpha$ times (in expectation) before it returns a witness. Thus in total our extractor runs k times slower than P' returns acceptable proofs, as desired.

Chapter 4

Proofs in the Common Reference String model

4.1 Merging proofs

We first exhibit a construction for combining two proofs of knowledge into a single proof of knowledge, of length still k, the lengths of each of the original proofs.

Construction 4.1.1 (Merging Proofs). *Suppose we have a system to express computationally-sound proofs of statements of the form "M when started on state s_1 reaches state s_2 after t time steps" as claims of membership of $(T, x) \in L$, with T a fixed machine and $x = (M, s_1, s_2, t)$. We exhibit a construction to merge an appropriate pair of such proofs.*

Assume we have the following: a machine M, an integer t, a triple of memory states of M s_1, s_2, s_3, and a pair of proofs of knowledge p_1, p_2 that prove respectively that running M for t steps from memory state s_1 yields state s_2, and running M for a further t steps from memory state s_2 yields state s_3. We combine these into a proof of knowledge that running M for $2t$ steps from memory state s_1 yields memory state s_3.

Explicitly we define a new pair (T', x') and claim that a proof of knowledge that $(T', x') \in L$ implies this combined statement.

Let $x' = (M, s_1, s_3, 2t)$. Define T' as a machine that on input (x', w') does the

following:

- *Interprets* x' *as the quadruple* $(M, s_1, s_3, 2t)$
- *Interprets* w' *as the triple* (p_1, p_2, s_2)
- *Verifies the proofs of knowledge* p_1, p_2 *respectively show that* $(T, (M, s_1, s_2, t))$ *and* $(T, (M, s_2, s_3, t))$ *are in the language* L*, namely checks that* $U(p_1, T, (M, s_1, s_2, t), r) = U(p_2, T, (M, s_2, s_3, t)) = 1$.

Since the w' *above is a witness that* $(T', x') \in L$*, we construct a proof of knowledge* $p' = P(T', x', w', r')$ *of this fact. We have thus constructed* x', p' *from the original* x_1, x_2, p_1, p_2*. (For reasons that will become clear later the random string* r' *used for this second-level proof should be independent of the random string* r *used for* p_1 *and* p_2*.)*

We now prove that any valid proof p' for the parameters $x' = (M, s_1, s_3, 2t)$ and the Turing machine T implies in a computationally-sound sense that running M for $2t$ time steps from state s_1 will reach state s_3. Call a pair $(x = (M, s_1, s_2, t), p)$ *deceptive* if p proves to the verifier that $(T, x) \in L$ but it is not the case that running M for t steps from memory state s_1 reaches memory state s_2.

Claim 4.1.2. *Suppose that* T *has the property that the probability that a machine running in time* b*,* $K' > b > 2t$*, outputs a* deceptive *pair* $((M, s_1, s_2, t), p)$ *is at most* $1/2$ *over the random string* r*. Then for any machine running in time* $\frac{1}{2}b/k^{c_2}$*, the probability that it outputs a deceptive pair for the second-level machine* T' *is less than* $1/K$.

Proof. This result is a straightforward consequence of the *knowledge extraction* property of the proofs in definition 2.3.1. Assume we have a machine X that outputs deceptive pairs $(x' = (M, s_1, s_3, 2t), p')$ for T' with probability $1/K$ (over r) in time $\frac{1}{2}b/k^{c_2}$. We apply the *extractor* E, and have by definition that $E(P', M, (s_1, s_3, 2t))$ returns a classical witness w' (relative to r') with probability at least $1/2$ in time at most $b/2$. The witness w' is a classical witness for (T', x') in the language L, and thus w' may be interpreted as $w' = (p_1, p_2, s_2)$ as in the construction, where, since w' is a classical witness, both the proofs p_1 and p_2 are accepted by the verifier. However, since p' is deceptive, at least one of p_1, p_2 must be deceptive (with respect to T). In

time $t < b/2$ we can classically check which one of p_1, p_2 is deceptive, by just simulating M for t steps on the appropriate input s_1 or s_2. Thus using $b/2 + b/2 = b$ time we have recovered a deceptive pair for T with probability at least $1/2$, which implies the desired result. $\square$

We have shown how to combine a pair of proofs into a single proof. We will recursively apply this construction to incrementally combine an arbitrary number of proofs of knowledge. A base case for the recursion is easily obtained:

Construction 4.1.3 (Base Case). *Let T_0 be the machine that operates on pairs (x, w) where $x = (M, s_1, s_2, 1)$, and checks that M when simulated for one step on configuration s_1 ends up in configuration s_2, ignoring the auxiliary input w.*

We thus construct a sequence of machines T_i, where machine T_i is used to verify claims where the simulation time t is 2^i, using random string r_i. The random string r of length k^2 of the incrementally verifiable computation scheme will be interpreted as the concatenation of strings $(r_0, r_1, ..., r_{k-1})$ to be used in the recursive construction just outlined.

We note that each witness w used in the recursive construction consists of a pair of proofs of knowledge (p_1, p_2), and a memory state (s_2) of M, and is thus of size $3k$. Further, merging two proofs via the construction involves constructing a single new proof of knowledge; this construction takes time polynomial in k by definition 2.3.1. Also, if we wish to prove statements about t steps of the computation of M, this will involve $\log t$ layers of the recursion. This will lower the security time bound from K' to $K'/(2 \cdot k^{c_2})^{\log t}$.

4.2 Incrementally verifying computation

We combine the observations of the previous section into the following result:

Theorem 4.2.1. *Given a noninteractive CS proof of knowledge (P, U, K', c, c_1, c_2), there exists an incrementally verifiable computation scheme (C, V, K) provided $k^{-\log k} K'/(2 \cdot k^{c_2})^{\log K} > K$.*

Proof. The above two constructions describe a recursive procedure for generating a proof of t steps of the computation using $\log t$ levels of a binary recursion. Consider the tree that such a recursion would induce. The leaves of the recursive tree are the memory configurations of M, and the internal nodes i levels above the leaves are proofs of knowledge of recursive depth i, asserting the results of simulating M for 2^i steps. Each node is computable in time polynomial in k from its two children, by Construction 4.1.1.

Consider a depth-first traversal of the binary tree, starting at the leaf corresponding to time 0 and visiting each leaf in order, computing the value of every node we visit. At any moment in such a traversal the "stack" consists of the values of nodes on a path from a leaf to the root. Every time a leaf is visited, we output the values of all the nodes along this path as a *proof of incremental correctness.* We note that processing any node takes time polynomial in k, and the depth of the recursion is less than k, and so a leaf is visited every $k^{O(1)}$ time. Thus this procedure uses the desired time and space.

We now show that these "stack dumps" in fact constitute computationally-sound proofs.

Consider a subtree whose leaves consist of a range $[t_1, t_2]$. When the recursion finishes processing that subtree, it will store in the parent node parameters $x = (M, s_1, s_2, t_2 - t_1 + 1)$ and a proof of knowledge that M when starting in configuration s_1 reaches configuration s_2 in time $t_2 - t_1 + 1$.

We note that when the recursion processes leaf t' it must have finished processing all the leaves before t', and thus the leaves spanned by those subtrees in the "stack" must constitute all the leaves before t'. Thus these proofs of knowledge, when considered together, assert the complete result of simulating M from time 0 to time t'.

To check such a sequence of proofs, we verify their individual correctness, and check that the start and end memory states for each of the corresponding "theorems" match up.

We note, as above, that if such a sequence of proofs is *deceptive*, then we can

(classically) isolate the deceptive proof in time $O(t)$ by simulating M. Provided this is less than K', the probability of fooling the verifier is thus bounded by repeated application of Claim 4.1.2 to be at most $1/K$. Thus if $k^{-\log k} K'/(2 \cdot k^{c_2})^{\log K} > K$ then our proof system is computationally sound.

We have thus transformed an arbitrary machine M into an incrementally-verifiable version of itself, as desired. □

Bibliography

[1] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[2] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, January 1998.

[3] B. Barak and O. Goldreich. Universal Arguments. *Proc. Complexity* (CCC) 2002.

[4] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. *STOC 2004, pp. 1-10.*

[5] M. Blum, A. De Santis, S. Micali, G. Persiano. Noninteractive Zero-Knowledge. SIAM J. Comput. *20(6): 1084-1118 1991.*

[6] M. Blum, P. Feldman, S. Micali. Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract). STOC 1988, pp. 103-112.

[7] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited, *STOC 1998*, pp. 209-218.

[8] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. *Advances in Cryptology* 2005.

[9] O. Goldreich and M. Sudan. Locally testable codes and PCPs of almost-linear length. *FOCS 2002.*

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

[11] J. Kilian. A note on efficient zero-knowledge proofs and arguments. *STOC*, 1992, pp. 723-732.

[12] S. Micali. Computationally Sound Proofs. *SIAM J. Computing* 30(4), 2000, pp. 1253–1298.

[13] R. Pass. On deniability in the common reference string and random oracle model. *Advances in Cryptology*, 2003, pp. 316-337.