comput. complex. **19** (2010), 265–304
1016-3328/10/020265-40, *published online* 14 May 2010
DOI 10.1007/s00037-010-0291-3

computational complexity

ARE PCPS INHERENT IN EFFICIENT ARGUMENTS?

Guy N. Rothblum and Salil Vadhan

Abstract. Starting with Kilian (STOC '92), several works have shown how to use probabilistically checkable proofs (PCPs) and cryptographic primitives such as collision-resistant hashing to construct very efficient argument systems (a.k.a. computationally sound proofs), for example with polylogarithmic communication complexity. Ishai et al. (CCC '07) raised the question of whether PCPs are inherent in efficient arguments, and if so, to what extent. We give evidence that they are, by showing how to convert any argument system whose soundness is reducible to the security of some cryptographic primitive into a PCP system whose efficiency is related to that of the argument system and the reduction (under certain complexity assumptions).

Keywords. Probabilistically checkable proof, computationally sound proof, argument, cryptographic reductions.

Subject classification. 68Q05.

1. Introduction

Probabilistically checkable proofs (PCPs) are one of the greatest successes of the interaction between complexity theory and the foundations of cryptography. The model of PCPs, and the equivalent model of multi-prover interactive proofs, emerged from efforts to find unconditional constructions of zero-knowledge proofs [8] and secure multiparty computation protocols [9, 15] (replacing the constructions of [25] and [44, 26], which relied on computational complexity assumptions). But like their predecessor, interactive proofs, they turned out to be extremely interesting from a purely complexity-theoretic point of view [19], particularly through their surprising connection to the inapproximability of optimization problems [18]. The PCP Theorem [3, 2] is one of the most celebrated results in complexity theory, and has led to a large body of work that continues to generate deep insights.

The PCP Theorem has also provided some returns to cryptography. Specifically, Kilian [33] showed how to use PCPs to construct arguments (i.e. computationally sound proof systems) for $\mathcal{NP}$ in which the communication complexity is polylogarithmic.[1] Kilian's construction assumes the existence of collision-resistant hash functions with subexponential security. Its zero-knowledge version [33] and other variants due to Micali [36] and Barak and Goldreich [6], have found further applications in cryptography [13, 5]. Moreover, these argument systems provide the asymptotically most efficient approaches for proving general $\mathcal{NP}$ statements, and thus are appealing for applications such as proving the correctness of a delegated computation or the safety of a program.

In this paper, we consider the question of whether PCPs are really necessary for very efficient arguments. One of our motivations is simply to better understand the relation between these two fundamental notions in complexity theory and cryptography. In addition, the use of PCPs in efficient argument systems has the drawback that the protocols and their applications inherit the somewhat complex construction and proof of the PCP Theorem. While there have been some substantial advances on simplifying the PCP Theorem [10, 17], it remains quite nontrivial and the construction may still be too involved to use in practice.

The question we study here has previously been considered by Ishai, Kushilevitz and Ostrovsky [31]. They showed that by using a stronger cryptographic primitive, namely (additively) homomorphic encryption rather than collision-resistant hashing, it is possible to construct somewhat efficient arguments using the simpler, exponential-length "Hadamard PCP" [2] rather than the polynomial-length PCPs of the full PCP Theorem. Their arguments are only "somewhat efficient" in that they have low (e.g. polylogarithmic) communication from the prover to the verifier, but the verifier-to-prover communication is polynomial (cf. [27]).

Our results. In this paper, we provide results suggesting that PCPs *are* necessary for constructing efficient arguments. Specifically, we consider a construction of an argument system based on a wide range of cryptographic primitives (e.g. collision-resistant hashing, the RSA assumption, homomorphic encryption), where the computational soundness is based on the security of the primitive via an efficient reduction. That is, there is an algorithm $\mathcal{S}$ such that if $\mathcal{P}^*$ is any prover strategy that convinces the verifier to accept a false statement, then

[1] Another important parameter is the computation time of the verifier, but we omit discussion of it in the introduction for the sake of simplicity.

$\mathcal{S}^{\mathcal{P}^*}$ "breaks" the cryptographic primitive.[2] For example, the cryptographic primitive could be a one-way function, and the reduction guarantees that any cheating prover that breaks soundness can be used to invert the function. Indeed, we provide a general formulation of a cryptographic primitive and what it means to "break" such a primitive. This formulation is quite general, covering standard primitives such as one-way functions or homomorphic encryption, and specific assumptions such as the hardness of factoring. For such constructions we show how to construct PCPs whose efficiency is related to that of the argument system, the reduction, and a variety of methods for "implementing" the cryptographic primitive (discussed more below).

Informally, our construction works as follows. We view the PCP oracle as specifying a prover strategy $\mathcal{P}_{\mathrm{PCP}}$ for the argument system (i.e. the next-message function). The PCP verifier:

1. Chooses an "implementation" C of the cryptographic primitive (to be discussed more below) and sends it to $\mathcal{P}_{\mathrm{PCP}}$ (with every query),

2. Runs the verifier of the argument system with $\mathcal{P}_{\mathrm{PCP}}$,

3. Runs the reduction $\mathcal{S}$ with $\mathcal{P}_{\mathrm{PCP}}$, and

4. Accepts if both the verifier of the argument system accepts (in Step 2) and $\mathcal{S}$ does not break the cryptographic primitive (in Step 3).

To establish soundness, we note that if $\mathcal{P}_{\mathrm{PCP}}$ convinces the verifier of the argument system of a false statement, then $\mathcal{S}$ will break the cryptographic primitive; this means that at least one of the acceptance conditions will fail. Thus, soundness of the PCP holds information-theoretically, unconditionally and regardless of the implementation chosen in Step 1 above. For completeness, we need to ensure that the implementation (of the cryptographic primitive) chosen in Step 1 cannot be broken by $\mathcal{S}^{\mathcal{P}}$, where $\mathcal{P}$ is the *honest* prover. We provide several methods for achieving this, some based on complexity assumptions and some unconditional. Below we describe a few of these and the resulting PCP parameters.

Implementation and parameters. Arguments and PCPs have many parameters, which we treat with as much generality as possible. But for simplicity in the current discussion, we focus on a few parameters of interest, with typical

[2] Thus, we require that the reduction is "black-box" in its access to $\mathcal{P}^*$, which is true of all of the existing constructions [33, 36, 6, 31].

settings. In particular, we ignore prover and verifier computation time, and the amount of randomness used by the verifier (see the discussion following Definition 2.2). Consider an argument system constructed from a cryptographic primitive C for a language $\mathcal{L} \in \mathcal{NP}$ such that on inputs of length n, the argument system has *prover-to-verifier* communication complexity polylog(n) and completeness and soundness error 1/3. Moreover, there is a poly(n)-time reduction $\mathcal{S}$ such that for every $x \notin \mathcal{L}$ and every cheating prover $\mathcal{P}^*$, if $\mathcal{P}^*$ convinces the verifier to accept with probability at least 1/3, then $\mathcal{S}^{\mathcal{P}^*}(C, x)$ "breaks" C with constant probability.

Three key parameters for us are the *verifier-to-prover* communication complexity $v = v(n)$; the number of rounds $r = r(n)$; and the number of queries $\mathcal{S}$ makes to its oracle $q = q(n)$. Kilian's construction of arguments from collision-resistant hash functions (CRHFs) of exponential hardness [33] achieves $v = O(\log n + \kappa)$, where κ is the seed length for a CRHF, and $r, q = O(1)$ (here we augment Kilian's construction by basing it on a PCP with constant query complexity, e.g. [3, 2]). The Ishai et al. [31] construction from homomorphic encryption has $v = \text{poly}(n)$ and $r, q = O(1)$.

Given the above, our resulting PCPs for $\mathcal{L}$ will always have constant completeness and soundness errors. The query complexity of our PCPs will be $r + q$, matching the known constructions of arguments from PCPs. Taking C to be the description length of the implementation of the cryptographic primitive generated by the verifier, the length of our PCPs is $2^{|C|+v}$. If $|C| = O(v)$, then this matches known constructions of arguments from PCPs in terms of the PCP length. I.e. if $|C| = O(v)$, then the PCP obtained is of comparable length (up to polynomial factors) as the one used in known constructions. For example, exponential-length PCPs correspond to $v = \text{poly}(n)$, and polynomial length PCPs correspond to $v = O(\log n)$.

Note that we use here the fact that the number of oracle queries made by $\mathcal{S}$ is small. This is the case in known constructions of argument systems, but one could certainly conceive of (or, given our results, hope for) a reduction that makes more queries and yields a PCP with query complexity that is too high to be interesting. Indeed, this is a promising avenue for future research, see the discussion below and in the appendix.

In this paper we present several approaches for implementing the cryptographic primitive used by the construction. Recall that our PCP verifier needs to generate an implementation C of the cryptographic primitive that cannot be broken by $\mathcal{S}^{\mathcal{P}}$, where $\mathcal{P}$ is the *honest* prover of the argument system. This is done in a variety of different ways, we outline a few below:

- ◦ The general framework above is formalized in Section 4, where we also present a natural (and efficient) instantiation. Assume that we have a secure implementation of the cryptographic primitive in the usual sense, e.g. that collision-resistant hash functions exist or that homomorphic encryption schemes exist, with whatever security parameter is used by the underlying argument system (typically polylog(n) to achieve polylogarithmic communication) and security against poly(n)-time adversaries. Such a primitive cannot be broken by $\mathcal{S}^{\mathcal{P}}$, because this is a poly$(n)$-time algorithm. Then, since the implementation C is described by a fixed algorithm (which gets a security parameter as input), it can be hardwired into the PCP protocol.

Here we obtain a standard (information-theoretically sound) PCP, where completeness relies on the assumption that the implementation of the primitive is secure. We view the assumption we use here as quite natural, as if there were no secure implementation of the primitive, then the original construction was a significantly less interesting object to begin with.

○ In Section 5, we consider more restrictive notions of reductions, where all entities in the argument system are given only black-box access to a cryptographic primitive such as a one-way function, pseudorandom function family, or collision resistant hash family. We show that, in some cases, we can *minimize* or *remove altogether* the computational assumptions made in the results presented above (for such fully black-box reductions). To do so, we observe that the implementation of the cryptographic primitive we use need only be secure against $\mathcal{S}^{\mathcal{P}}$: a single fixed polynomial-time algorithm. Obtaining an implementation that is secure against a fixed polynomial time bounded algorithm can be considerably easier than obtaining an implementation secure against *any* polynomial-time algorithm (the usual requirement from cryptographic primitives).

For example, we can obtain *unconditional* implementations of collision-resistant hash functions against $\mathcal{S}^{\mathcal{P}}$ using a family of poly(n)-wise independent hash functions, yielding $|C| = \text{poly}(n)$. Plugging these into our construction above we obtain exponential-length PCPs. The PCP length can be made independent of $|C|$ by viewing it instead as a "randomized PCP" (where the prover and verifier have shared randomness, which we use to generate C)[3], or by derandomizing the construction of

[3] Recalling that standard PCPs can be viewed as Karp reductions to approximate constraint satisfaction problems, these randomized PCPs can be viewed as randomized Karp reductions to approximate constraint satisfaction problems. Later in this work we refer to such randomized PCPs as "AM-PCPs".

C using either nonuniformity, or the pseudorandom generator of [30] (the latter being under the assumption that $\mathcal{EXP} = \mathcal{DTIME}(2^{O(n)})$ requires circuits of size $2^{\Omega(n)}$.).

Remarks. We emphasize that even though we use complexity assumptions in some of our transformations, the resulting PCPs achieve the standard, statistical definition of soundness — there does not exist *any* proof oracle that can convince the verifier to accept a false statement, except with small probability. Indeed, the complexity assumptions are only used for the *completeness* of (some of) our constructions, in order to ensure that the honest proof oracle that describes the prover does not inadvertently allow (the reduction $\mathcal{S}$) to break the primitive. This conditional completeness differs from the soundness of the original argument system, which also held under the same assumption, but was only guaranteed against bounded malicious provers.

We also note that the main objects we study, PCPs and computationally sound argument systems, give incomparable soundness guarantees. PCPs guarantee soundness against computationally unbounded but non-adaptive malicious provers (i.e. a provers whose answers are not allowed to depend on queries that the verifier made previously). Argument systems guarantee soundness against computationally bounded (e.g. polynomial-time), but fully adaptive malicious provers. Nonetheless, all our results apply even for constructions of *computationally sound PCPs*, i.e. proof systems with soundness that is only guaranteed against computationally bounded *and* non-adaptive provers. In particular, we conclude that any construction of a computationally sound PCP whose soundness is efficiently black-box-reducible to some cryptographic primitive can be converted into an information-theoretically sound PCP (whose completeness, as above, may rely on assumptions).

Perspective. Like all negative results regarding reductions, our results do not entirely preclude the possibility of obtaining efficient arguments "without PCPs," and may be alternatively interpreted as suggesting avenues for doing so. One possibility is to make non-black-box use of the cheating prover strategy $\mathcal{P}^*$ in the reduction from breaking the cryptographic primitive to violating soundness (or at least to use the fact that $\mathcal{P}^*$ is efficient). Another is to make use of reductions $\mathcal{S}$ that make many queries q to the cheating prover, even when soundness is violated with constant probability. (If $q = \text{poly}(n)$, we get a PCP with $\text{poly}(n)$ queries, which is trivial.) Another direction is to use a reduction where a malicious prover that breaks soundness with even say constant probability only breaks the cryptographic primitive used with polynomially small advantage. Such a reduction would also yield only a PCP with polynomial query complexity. Known reductions are not of any of the above types.

2. Preliminaries and definitions

Let $[n]$ be the set $\{1, 2, \ldots n\}$. For $x, y \in \{0,1\}^n$ we use $x \circ y$ to denote the concatenation of x and y (a string in $\{0,1\}^{2n}$). For a (discrete) distribution D over a set X we denote by $x \sim D$ the experiment of selecting $x \in X$ by the distribution D. A function $f(n)$ is *negligible* if it is smaller than any (inverse) polynomial. We refer the reader to [20, 21] for complete definitions of standard cryptographic objects used in this work such as one-way functions, distribution ensembles, collision resistant hash functions and pseudorandom functions. We emphasize that throughout this work whenever we make or refer to hardness assumptions, the assumptions are always against nonuniform adversaries.

We present definitions of the two types of proof system we consider in this work:

Definition 2.1 (Argument system $(\mathcal{P}, \mathcal{V})$ [12]). *An argument system for a language $\mathcal{L} \in \mathcal{NTIME}(g(n))$ consists of two interactive machines. $\mathcal{P}(x, w)$ gets an n-bit input x and advice $w \in \{0,1\}^{g(|x|)}$ (usually a witness to the input's membership in $\mathcal{L}$). $\mathcal{V}(x)$ gets the input x. The requirements are:*

- *Completeness $c(n)$. For every $x \in \mathcal{L}$ and corresponding witness w for x's membership in $\mathcal{L}$, the interaction of $\mathcal{V}(x)$ with $\mathcal{P}(x, w)$ (sometimes denoted $(\mathcal{P}(x, w), \mathcal{V}(x))$) makes $\mathcal{V}$ accept with probability at least $c(n)$.*
- *Soundness $s(n)$ vs. size $f(n)$. For every n-bit input $x \notin \mathcal{L}$ and every cheating $\mathcal{P}^*$ of (nonuniform) circuit size at most $f(n)$, the probability that $(\mathcal{P}^*(x), \mathcal{V}(x))$ makes $\mathcal{V}$ accept is at most $s(n)$.*

There are many complexity measures of an argument system that will interest us, such as the communication complexity (in each direction), the round complexity, the circuit size of the honest prover and verifier, and more.

Definition 2.2 (PCP $(\mathcal{P}, \mathcal{V})$ [19, 4, 3]). *A Probabilistically Checkable Proof (PCP) for a language $\mathcal{L} \in \mathcal{NTIME}(g(n))$ consists of a non-adaptive (i.e. stateless) machine $\mathcal{P}$ and an oracle machine $\mathcal{V}$. $\mathcal{P}(x, w)$ gets an n-bit input x, advice $w \in \{0,1\}^{g(|x|)}$ (usually a witness to the input's membership in $\mathcal{L}$) and an input oracle query. $\mathcal{V}(x)$ gets the input x. The requirements are:*

- ○ *Completeness $c(n)$. For every $x \in \mathcal{L}$ and corresponding witness w for x's membership in $\mathcal{L}$, the probability that $\mathcal{V}(x)$ accepts when it is run with $\mathcal{P}(x, w)$ as its oracle (we denote this as $\mathcal{V}(x)^{\mathcal{P}(x,w)}$), is at least $c(n)$.*

- ○ *Soundness $s(n)$. For every n-bit input $x \notin \mathcal{L}$ and every non-adaptive cheating $\mathcal{P}^*$ oracle, the probability that $\mathcal{V}(x)^{\mathcal{P}^*(x)}$ accepts is at most $s(n)$.*

There are many complexity measures of an PCP system that have been extensively studied. In this work we focus on the query complexity (the number of oracle calls $\mathcal{V}$ makes), the alphabet size (the size of $\mathcal{P}$'s output), the PCP length (the number of possible $\mathcal{P}$ input queries), the circuit size of the honest prover and verifier, and more.

Note that, unlike much of the literature, we do not focus on the PCP verifier's randomness complexity, but this quantity is closely related to proof length. In general, the randomness complexity of the verifier provides an (exponential) upper bound on the proof length. In the other direction, for "randomized PCPs" (or "AM-PCPs", see Section 5.1), where the prover and the verifier are allowed to share common randomness, the verifier's randomness can always be reduced to be logarithmic in the proof length. (See also [22, Exercises 9.15,9.16].)

3. Cryptographic primitives and reductions

In this section we consider reductions from computationally sound argument systems to cryptographic primitives (we use the terminology of Reingold, Trevisan and Vadhan [40], where a reduction *from A to B* means that the existence of B implies the existence of A). We would like to consider a general notion of a cryptographic primitive and of a reduction.

We begin with intuition. In the formalism below, a cryptographic primitive is associated with a class of circuits implementing the primitive and a *testing* procedure that tests whether a given adversary *breaks* the primitive. For example, we can illustrate the formalism via one-way functions. Here the circuit family implementing the primitive is the family of circuits computing the function. The testing procedure, to test a given adversary, repeatedly generates a random input, computes the function's output on it, and feeds the output to the adversary. The testing procedure accepts if the adversary successfully inverts the function in one of its invocations. This notion, presented in Definition 3.1, captures a host of cryptographic primitives such as one-way functions, encryption schemes, specific assumptions, and more. See below for a full discussion regarding how they fit the formalism.

Definition 3.1 (Cryptographic primitive). *A* cryptographic primitive $(\mathcal{C}, \mathcal{T})$ *is defined by a class* $\mathcal{C}$ *of circuits and a* testing *procedure* $\mathcal{T}$*. For a candidate* C *in* $\mathcal{C}$ *(a circuit in the class), we say that an interactive adversary* $\mathcal{A}$ (κ, ε)-breaks C *if:*

$$\Pr_{\mathcal{T}\text{'s coins}}\left[\mathcal{T}^{\mathcal{A}}(C, 1^{\kappa}, 1^{\lceil 1/\varepsilon \rceil}) \textit{ accepts }\right] \geq 2/3\,.$$

On the other hand, we say C is (κ, ε)-secure *against* $\mathcal{A}$ *if:*

$$\Pr_{\mathcal{T}\text{'s coins}}\left[\mathcal{T}^{\mathcal{A}}(C, 1^{\kappa}, 1^{\lceil 1/\varepsilon \rceil}) \textit{ accepts }\right] \leq 1/3$$

where in both cases $\mathcal{T}$ *is given access to the circuit* C*. Throughout this work we deal only with* C *and* $\mathcal{A}$ *for which there is a* promise *that either* $\mathcal{A}$ *breaks* C*, or* C *is secure against* $\mathcal{A}$*. The input parameter* κ *is typically used to denote a security parameter that bounds the input and output sizes of circuit* C *(circuits that don't meet this bound make* $\mathcal{T}(C, 1^{\kappa}, \cdot)$ *accept immediately).*[4] *Intuitively, the parameter* ε *is used to specify a "threshold" for the success probability of* $\mathcal{A}$ *in breaking the primitive, see the examples below.*

Note that the above notion can be extended to consider classes of circuit distributions (rather than circuits, or circuit distributions with support size 1, as done above). For simplicity and clarity we use the more restricted notion (Definition 3.1 suffices to capture all the cryptographic primitives we consider in this work). We proceed by considering several examples and how they fit into the above definition of a cryptographic primitive:

1. One-Way Functions. Here $\mathcal{C}$ is the class of circuits computing a function, say from $\{0,1\}^{\kappa}$ to $\{0,1\}^{\kappa}$. Given a circuit C and adversary $\mathcal{A}$, the tester $\mathcal{T}^{\mathcal{A}}(C, 1^{\kappa}, 1^{\lceil 1/\varepsilon \rceil})$ chooses $O(1/\varepsilon)$ random inputs to the function, applies C to each of the inputs, and runs $\mathcal{A}$ on each of the outputs. $\mathcal{T}$ accepts if the adversary inverts C on at least one of these outputs (i.e. $C(A(C(x))) = C(x)$ for one of the inputs x). If $f : \{0,1\}^* \to \{0,1\}^*$ is a (length-preserving) one-way function, this means that it is computable in polynomial time (in its input length), and for every PPT $\mathcal{A}$ and polynomial $p(\cdot)$, for sufficiently large κ, f_{κ} is $(\kappa, 1/p(\kappa))$-secure against $\mathcal{A}_{\kappa}$. I.e., it holds that: $\Pr[\mathcal{T}^{\mathcal{A}_{\kappa}}(f_{\kappa}, 1^{\kappa}, 1^{p(\kappa)})$ accepts $] \leq 1/3$, where f_{κ} and $\mathcal{A}_{\kappa}$ are the restrictions of f and $\mathcal{A}$ to inputs of length κ.

   We can also consider subexponentially-hard one-way functions. If $f : \{0,1\}^* \to \{0,1\}^*$ is a (length-preserving) subexponentially-hard one-way

[4] Note that even though κ could also be used to bound the circuit size of the circuit C, we will not do so in this work.

function, then it is computable in polynomial time (in its input length), and for some constant $\delta > 0$ and every probabilistic algorithm $\mathcal{A}$ running in time at most 2^{κ^δ}, for sufficiently large κ, f_κ is $(\kappa, 1/2^{\kappa^\delta})$-secure against $\mathcal{A}_\kappa$.

2. Collision-Resistant Hash Families. Here $\mathcal{C}$ is the class of circuits that evaluate families of shrinking hash functions say from $\{0,1\}^{2\kappa}$ to $\{0,1\}^\kappa$. I.e., C in $\mathcal{C}$ gets as input a seed s and an input x and outputs the function $C_s(x) = C(s,x)$. The tester $\mathcal{T}$ chooses $O(1/\varepsilon)$ random seeds $\{s_1, s_2, \ldots, s_{O(1/\varepsilon)}\}$, and asks $\mathcal{A}$ to find a collision on each of them. It accepts if $\mathcal{A}$ succeeds on at least one (i.e. if for any of the seeds s_i, the adversary $\mathcal{A}(s_i)$ finds x and x' such that $C(s_i, x) = C(s_i, x')$).

3. Hardness of Factoring. We can also view *specific* number-theoretic (or other) assumptions as cryptographic primitives in our framework. To capture, for example, the assumption that factoring is hard, we have a single circuit C_κ for every value of the security parameter. This circuit C_κ is the canonical circuit that picks two random $\kappa/2$-bit primes and outputs their product. The tester $\mathcal{T}$ takes $O(1/\varepsilon)$ random samples (numbers) $\{n_1, n_2, \ldots, n_{O(1/\varepsilon)}\}$ from the distribution. It then asks $\mathcal{A}$ to factor each of these numbers, and accepts if $\mathcal{A}$ succeeds on at least one ($\mathcal{A}$ finds a non-trivial factorization of some n_i into two prime factors). Alternatively, the circuit C_κ could be empty, with the tester $\mathcal{T}$ generating the primes and their products on its own.

4. Homomorphic Encryption Scheme. A homomorphic encryption scheme is a (public or secret key) scheme with a special homomorphic evaluation procedure that can be used on a sequence of ciphertexts to compute an encryption of some function f of the plaintexts (common functions include addition and multiplication). The scheme remains semantically secure against an adversary who is given a circuit computing this homomorphic evaluation procedure. See [21] for more details on semantically secure encryption schemes.

   In this example, $\mathcal{C}$ is the class of circuits that perform key generation, encryption, decryption and homomorphic evaluation procedures. The tester uses $C \in \mathcal{C}$ to generate a key and feeds the adversary with the homomorphic evaluation procedure (for public-key schemes, the adversary is also given the encryption procedure). The tester and adversary then run the semantic security game $O(1/\varepsilon^2)$ times, and the tester accepts if the adversary has advantage ε in breaking the scheme's semantic security in these experiments (to detect w.h.p. an ε-advantage we need to run the experiment $O(1/\varepsilon^2)$ times).

Discussion. Note that the definition of a cryptographic primitive is decoupled from the question of whether there exists an *implementation* of the primitive that is (κ, ε)-secure against a collection of adversaries. The related work of Naor also considers general notions of cryptographic assumptions and primitives [38]. That work sets forth the notion of *falsifiable* assumptions: assumptions for which there exists a procedure for testing whether a given adversary breaks the assumption. The primary focus there is classifying cryptographic assumptions according to how efficiently they can be *falsified*. In that setting, one of the goals is designing specific procedures that not only break the cryptographic assumption (assuming that it can be broken), but that do so in a way that can be verified very efficiently. In our notion of a cryptographic primitive, we consider falsifiable primitives, but we focus on verifying that an *arbitrary* adversary (provided by a security reduction) breaks the cryptographic primitive. Falsifiable (and even only somewhat falsifiable, cf. [38] assumptions) naturally fall into our framework of a cryptographic primitive. Non-falsifiable assumptions, such as the knowledge of exponent assumption [16], may not fit into our notion. This is because there is no efficient "testing procedure" that can be used to tell whether an adversary breaks the assumption; we need the testing procedure to be efficient because it is run by the (efficient) PCP verifier.

Haitner and Holenstein [29] also consider reductions to a wide class of cryptographic primitives (the reductions are from encryption schemes with key-dependant security). Their notion of a "cryptographic game" does not distinguish between the testing procedure and the description of the primitive.

Now that we have presented our notion of a cryptographic primitive, we proceed to define a *reduction* from a computationally sound argument system to a cryptographic primitive.

Definition 3.2 (Reduction). *A reduction $\mathcal{R} = (\mathcal{P}, \mathcal{V}, \mathcal{S}, (\mathcal{C}, \mathcal{T}), \kappa(\cdot), \varepsilon(\cdot))$ from an argument system for a language $\mathcal{L}$ to a cryptographic primitive defined by $(\mathcal{C}, \mathcal{T})$, consists of several components. We use x to denote an n bit input whose membership is being proved, and w to denote the prover's auxiliary input (usually a witness to the input's membership).*

(i) A cryptographic primitive $(\mathcal{C}, \mathcal{T})$ as in Definition 3.1.

(ii) Two functions $\kappa : \mathbb{N} \to \mathbb{N}$, $\varepsilon : \mathbb{N} \to [0, 1]$ that determine the parameters of the cryptographic primitive as a function of the input length.

The function $\kappa(\cdot)$ determines the security parameter, and $\varepsilon(\cdot)$ determines the advantage of an adversary who breaks the argument's soundness in breaking the cryptographic primitive.[5]

(iii) Two interactive Turing machines: a prover $\mathcal{P}(C, x, w)$ and verifier $\mathcal{V}(C, x)$ with access to a candidate circuit C in $\mathcal{C}$.

(iv) A proof of security: an oracle machine $\mathcal{S}$ with black-box access to a cheating prover $\mathcal{P}^*(C, x)$ that gets as input C in $\mathcal{C}$ and $x \in \{0,1\}^n$.

We require that $(\mathcal{P}(C,\cdot,\cdot), \mathcal{V}(C,\cdot))$ is complete for every *candidate $C \in \mathcal{C}$. For security, we require that if a cheating prover $\mathcal{P}^*(C, x)$ violates soundness for some $x \notin \mathcal{L}$ and C, then $\mathcal{S}^{\mathcal{P}^*}(\cdot, x)$ breaks the (supposedly hard) C. If C is indeed hard to break, then the argument system is thus sound. We state these requirements formally below:*

(i) Completeness $c(n)$. For every C in $\mathcal{C}$, given $x \in \mathcal{L}$ and a valid witness w, the prover $\mathcal{P}(C, x, w)$ convinces $\mathcal{V}(C, x)$ with probability at least $c(n)$.

(ii) Security proof of soundness $s(n)$. For every C in $\mathcal{C}$, every n-bit input $x \notin \mathcal{L}$ and every cheating prover $\mathcal{P}^*(C, x)$: if $(\mathcal{P}^*(C), \mathcal{V}(C))(x)$ accepts with probability at least $s(n)$, then $\mathcal{S}^{\mathcal{P}^*(\cdot,x)}$ breaks C, i.e.:

$$\Pr\left[\mathcal{T}^{\mathcal{S}^{\mathcal{P}^*(\cdot,x)}}(C, 1^{\kappa(n)}, 1^{\lceil 1/\varepsilon(n)\rceil}) \textit{ accepts}\right] \geq 2/3\,.$$

For simplicity, one can think of $\varepsilon(n) = s(n)^{O(1)}$ throughout this work.

We assume throughout that $s(n) \leq 0.1$ and $c(n) \geq 0.9$. We use $t(n)$ to denote the circuit size of $\mathcal{S}^{\mathcal{P}}$ (i.e., $t(n)$ is at most $|\mathcal{S}| \cdot |\mathcal{P}|$, here we refer only to the honest $\mathcal{P}$), and $q(n)$ to denote the number of $\mathcal{P}^*$-oracle queries made by $\mathcal{T}^{\mathcal{S}^{\mathcal{P}^*}}$.[6] *We use $v(n)$ to denote a bound on the number of bits sent from $\mathcal{V}$ to $\mathcal{P}$, $u(n)$ to denote a bound on the length (in bits) of each of $\mathcal{P}$'s answers, and $r(n)$ to denote the number of rounds of communication of $(\mathcal{P}, \mathcal{V})$.*

[5] We find it convenient to have the reduction determine the security parameter $\kappa = \kappa(n)$ and the advantage in breaking the cryptographic primitive $\varepsilon = \varepsilon(n)$, rather than give κ as input to all the algorithms.

[6] Throughout, whenever we refer to a bound on a parameter that depends on $\mathcal{P}^*$ we mean the worst case bound over the input, the cheating prover, etc. for input length n. Note that these bounds may also depend on the security parameter $\kappa(n)$, which is a parameter of the reduction.

In the reduction notion of Definition 3.2, all the algorithms in the argument system (prover, verifier, tester $\mathcal{T}$) get access to C's explicit representation. The only "black-box" access in the definition is the security proof's access to the cheating prover. This is quite a general notion of reduction. See [40] for a discussion of different notions of reductions. In this work we also consider more restricted notions. (Fully) black-box reductions are reductions where the algorithms access C as a black box. We will also consider black-box reductions with bounded adaptiveness. See Section 5 for a discussion and definitions of these more restricted types of reductions. See Appendix A for an overview of known argument constructions and a discussion of how they fit into our framework.

4. From arguments to PCPs

In this section, we take any reduction $\mathcal{R} = (\mathcal{P}, \mathcal{V}, \mathcal{S}, (\mathcal{C}, \mathcal{T}), \kappa(\cdot), \varepsilon(\cdot))$ from an argument system for a language $\mathcal{L}$ to a cryptographic primitive specified by $(\mathcal{C}, \mathcal{T})$, and construct from it a PCP for $\mathcal{L}$.

4.1. A generic transformation. For all of our results, we need an additional property from the reduction. We require that it is possible to generate candidates C in $\mathcal{C}$ for the cryptographic primitive, that cannot be broken by the security proof $\mathcal{S}^{\mathcal{P}}$ when it runs with the *honest* prover (except with small probability). We formalize this property below.

Property 4.1. *The reduction $\mathcal{R}$ (with soundness $s(n)$) has a (polynomial time deterministic) generation procedure $\mathcal{G}(1^n)$ that outputs a candidate C in $\mathcal{C}$ such that for every $x \in \mathcal{L}$ and every valid witness w for x's membership in $\mathcal{L}$, C is $(\kappa(n), \varepsilon(n))$-secure against $\mathcal{S}^{\mathcal{P}(\cdot,x,w)}(\cdot, x)$.*[7]

In Section 5 we extend this notion to probabilistic generators $\mathcal{G}$. We will restrict our attention to deterministic $\mathcal{G}$ throughout this section.

We now specify a "generic" PCP construction for reductions with Property 4.1. We will later show how to instantiate this generic construction for specific cryptographic primitives, by constructing a generator $\mathcal{G}$ that meets Property 4.1 (unconditionally or under various assumptions). Recall that we view the verifier for the PCP as an oracle machine (with oracle access to the proof or oracle-prover). We run the generator $\mathcal{G}$ to generate a candidate C.

[7] Recall that in Definition 3.1 we captured "security against $\mathcal{A}$" by saying that the testing procedure $\mathcal{T}$ accepts $\mathcal{A}$ with probability at most 1/3.

The generic verifier $\mathcal{V}_{\mathrm{PCP}}$ and the (honest) prover oracle $\mathcal{P}_{\mathrm{PCP}}$ depend on this candidate C.

Verifier $\mathcal{V}_{\mathrm{PCP}}(C, x)$.

1. Choose random coins for $\mathcal{V}$. Simulate $\mathcal{V}(C, x)$ in the interactive argument system using these coins and the candidate C, using the PCP prover $\mathcal{P}_{\mathrm{PCP}}$ to obtain the messages of the argument system's prover $\mathcal{P}(C, x, w)$.

   Thus, each query to $\mathcal{P}_{\mathrm{PCP}}$ specifies a transcript of the interactive argument (the verifier's messages are computed using the existing transcript and the random coins chosen.) If $\mathcal{V}$ rejects, then reject. Otherwise, continue to Step 2.

2. Repeat the following $O(\log(1/\alpha))$ times, where $\alpha = \alpha(n)$ is a parameter (its effect on completeness and soundness is analyzed below):

   Run the tester $\mathcal{T}^{\mathcal{S}^{\mathcal{P}_{\mathrm{PCP}}}}(C, 1^n, 1^{\lceil 1/\varepsilon(n) \rceil})$ with independent random coins to check whether $\mathcal{S}^{\mathcal{P}_{\mathrm{PCP}}}$ breaks C. Here $\mathcal{P}_{\mathrm{PCP}}$ plays the role of answering $\mathcal{S}$'s oracle queries to $\mathcal{P}^*$. Again, each query to $\mathcal{P}_{\mathrm{PCP}}$ specifies a transcript of the interactive argument.

   If in at least half of these iterations $\mathcal{T}$ accepts, then reject. Otherwise accept.

Figure 4.1: Verifier $\mathcal{V}_{\mathrm{PCP}}$.

(Honest) PCP Proof Oracle $\mathcal{P}_{\mathrm{PCP}}(C, x, w)$

For any query specifying a transcript of past messages for the interactive argument, simulate $\mathcal{P}(C, x, w)$ on this transcript and output its next message.

Figure 4.2: (Honest) PCP Proof Oracle $\mathcal{P}_{\mathrm{PCP}}$.

The intuition is that if for $x \notin \mathcal{L}$ a cheating PCP prover $\mathcal{P}^*_{\mathrm{PCP}}$ makes the verifier $\mathcal{V}_{\mathrm{PCP}}$ accept with probability $s(n)$ or greater in Step 1, then the reduction $\mathcal{R}$ guarantees that in Step 2, the security proof $\mathcal{S}^{\mathcal{P}^*_{\mathrm{PCP}}}$ will break C correctly with advantage $\varepsilon(n)$ (and $\mathcal{V}_{\mathrm{PCP}}$ rejects). This guarantees soundness. On the other hand, when $x \in \mathcal{L}$, we know by Property 4.1 that C is $(\kappa(n), \varepsilon(n))$-secure against the security proof run with the honest prover (and so the verifier should usually accept). This guarantees completeness. We formalize this in the theorem below.

Theorem 4.2. *Let $\mathcal{R} = (\mathcal{P}, \mathcal{V}, \mathcal{S}, (\mathcal{C}, \mathcal{T}), \kappa(\cdot), \varepsilon(\cdot))$ be a reduction from an argument system for a language $\mathcal{L}$ to a cryptographic primitive specified by $(\mathcal{C}, \mathcal{T})$, as in Definition 3.2. Suppose furthermore that $\mathcal{R}$ satisfies Property 4.1 and has a generator $\mathcal{G}$ for hard candidates.*

Let $c = c(n)$ and $s = s(n)$ be the completeness and soundness of the argument system, and take $\varepsilon = \varepsilon(n)$ and $\kappa = \kappa(n)$. Recall that $v = v(n)$ bounds the communication from $\mathcal{V}$ to $\mathcal{P}$, the value $u = u(n)$ bounds $\mathcal{P}$'s answer lengths, the value $r = r(n)$ bounds the number of rounds, and $q = q(n)$ bounds the number of $\mathcal{P}^*$-queries made by $\mathcal{T}^{\mathcal{S}^{\mathcal{P}^*}}(C, 1^n, 1^{\lceil 1/\varepsilon \rceil})$.

Then $(\mathcal{P}_{\mathrm{PCP}}, \mathcal{V}_{\mathrm{PCP}})$ is a PCP for $\mathcal{L}$ with completeness $c - \alpha$ and soundness $\max\{s, \alpha\}$. The number of queries is $r + O(\log(1/\alpha) \cdot q)$. The alphabet size is 2^u. The length of the PCP is 2^v. Furthermore, the PCP oracle can be evaluated in time polynomial in the running time of $\mathcal{P}(C, x, w)$. The running time of the PCP verifier is polynomial in the running time of $\mathcal{G}$, of $\mathcal{V}(C, x)$ and of $\mathcal{T}^{\mathcal{S}^{\mathcal{P}^*}}(C, 1^n, 1^{\lceil 1/\varepsilon \rceil})$.

Proof. We begin by analyzing the proposed construction's alphabet size, length and query number:

- Query number: The verifier $\mathcal{V}_{\mathrm{PCP}}$ makes r queries to $\mathcal{P}_{\mathrm{PCP}}$ in Step 1 (one for each round of communication between $\mathcal{V}$ and $\mathcal{P}$). It then runs $O(\log(1/\alpha))$ simulations of $\mathcal{S}$, each of which makes q queries. The total number of queries is thus $r + O(\log(1/\alpha) \cdot q)$.

- Alphabet size: The answers of $\mathcal{P}_{\mathrm{PCP}}$ are messages sent by the prover $\mathcal{P}$ in the interactive argument, their length is bounded by u and the alphabet size is bounded by 2^u.

- PCP length: Each query made by $\mathcal{V}_{\mathrm{PCP}}$ includes a transcript for the interactive argument. The length of each such query is thus v, and the length of the PCP is 2^v.

We now turn our attention to completeness and soundness. For soundness, suppose $x \notin \mathcal{L}$ but $\mathcal{P}^*_{\mathrm{PCP}}$ makes the verifier $\mathcal{V}_{\mathrm{PCP}}$ accept in Step 1 with probability at least s. We view $\mathcal{P}^*_{\mathrm{PCP}}$ as a cheating prover $\mathcal{P}^*$ for the interactive argument. By Property (ii) in Definition 3.2 (security proof of soundness), we know that $\mathcal{S}^{P^*}(\cdot, x)$ will have advantage ε in breaking C. This means that in Step 2, every time that $\mathcal{V}_{\mathrm{PCP}}$ simulates $\mathcal{T}$, it accepts with probability at least 2/3. Thus (repeating $\Theta(\log(1/\alpha))$ times), the verifier $\mathcal{V}_{\mathrm{PCP}}$ will reject in Step 2 with all but probability α. On the other hand, if the probability of "accepting" (i.e. not rejecting) in Step 1 is smaller than s, then the total probability of the verifier accepting is smaller than s. Hence, the total probability of accepting is at most $\max\{s, \alpha\}$.

For completeness, in Step 1 of $\mathcal{V}_{\text{PCP}}$'s operation, when it runs the argument system's $\mathcal{V}$, it will accept with probability at least c by the completeness of $(\mathcal{P}, \mathcal{V})$. By Property 4.1, the candidate C is (κ, ε)-secure against $\mathcal{S}^{\mathcal{P}_{\text{PCP}}}$. So in every iteration of Step 2, $\mathcal{T}$ will reject with probability at least 2/3. Repeating $O(\log(1/\alpha))$ times, the probability that the verifier rejects in Step 2 is at most α. Taking a union bound, the total probability of accepting when $x \in \mathcal{L}$ and the prover is honest is at least $c - \alpha$. □

4.2. Constructions under cryptographic assumptions. As an immediate corollary of Theorem 4.2, we obtain conditional constructions of PCPs from argument systems. If there is indeed a computationally hard candidate for the cryptographic primitive on which the argument's construction is based, then this candidate immediately satisfies Property 4.1. We view this as a natural assumption to make: presumably we consider the construction of an argument to be meaningful because we believe that the cryptographic primitive has a secure implementation. Given such an implementation, we get a PCP (with statistical soundness) "for free". We can use the candidate to construct the PCP. We emphasize that *the soundness of the PCP obtained is unconditional and information-theoretic*; it is only completeness that is based on the cryptographic assumption. In fact, it suffices that the implementation is secure against (the reduction run with) the *fixed polynomial-time bounded honest prover*, so we can even make do with a cryptographic primitive that is only secure against this fixed algorithm (we elaborate and build on this in subsequent sections).

Here the notion of reduction from arguments to cryptographic primitives used is the general notion of Definition 3.2, i.e. the reduction is black-box only in the adversary. In particular, we obtain the following (informal) corollary:

Corollary 4.3 (Informal). *Let $\mathcal{R}$ be a reduction from a computationally sound argument system for language $\mathcal{L}$ to a cryptographic primitive. If there exists a secure implementation of the cryptographic primitive, then the argument system can be used to construct a PCP as in Theorem 4.2.*

In particular, if there exists a family of collision-resistant hash functions, then any reduction from a computationally sound argument system to a CRHF can be used to construct a PCP. If there exists an additively homomorphic encryption scheme, then any reduction from a computationally sound argument system to additively homomorphic encryption can be used to construct a PCP.

Perspective from known constructions. We first examine the known reductions using collision-resistant hashing for $\mathcal{NP}$ arguments [33, 36, 6]. Taking κ to be the security parameter, the communication from $\mathcal{V}$ to $\mathcal{P}$ is $v(n) = O(\log n + \kappa)$ (specifying the hash function and $O(1)$ PCP queries), the length of prover answers is $u(n) = O(\kappa \cdot \log n)$, and the number of rounds is $r(n) = O(1)$. The number of calls $\mathcal{T}^{\mathcal{S}^{\mathcal{P}^*}}$ makes to $\mathcal{P}^*$ is $q(n) = O(1)$ (for constant soundness and advantage in breaking the primitive). Theorem 4.2 gives (for any instantiation) a PCP with constant completeness and soundness, $O(1)$ queries, alphabet size $2^{O(\kappa \cdot \log n)}$, and proof length $\text{poly}(n) \cdot 2^{\kappa}$. Thus, if we take a poly-logarithmic security parameter, the PCP length is quasi-polynomial. This does not quite match the Kilian [33] construction (which needed a polynomial-length PCP), but as we show in Section 5, we can actually (under complexity assumptions) get implementations of the CRHF that suffice for the construction above and with logarithmic κ. This yields a polynomial-length PCP from any (black-box) construction with the parameters of [33].

If we examine the reduction of [31], there the communication from the verifier to the prover is κ times the logarithm of the length of the PCP being used, $v(n) = \text{poly}(n) \cdot \kappa$ (in their case the PCP used was exponential, and so $v(n)$ is polynomial). The communication from the prover to the verifier is $u(n) = O(\kappa)$, and the number of rounds is $r(n) = O(1)$. Again, the number of calls $\mathcal{T}^{\mathcal{S}^{\mathcal{P}^*}}$ makes to $\mathcal{P}^*$ is $q(n) = O(1)$ (for constant soundness and advantage in breaking the encryption). Theorem 4.2 gives (for any instantiation) a PCP with constant completeness and soundness, $O(1)$ queries, alphabet size $2^{O(\kappa)}$, and proof length $2^{\text{poly}(n) \cdot \kappa}$ (as should be expected, because they started with an exponential length PCP).

5. Weakening or eliminating computational assumptions

In this section we consider more restricted reductions than those of Definition 3.2, and obtain PCP constructions with better parameters than the ones obtained in Section 4.2. The main idea will be to build an implementation for the cryptographic primitive used by the reduction that is only secure against one specific adversary: the adversary that runs the reduction together with the *honest* argument prover (such an implementation still suffices for arguing completeness via Theorem 4.2). In this section, we will look at *(fully) black-box reductions* and (even more restricted) *black-box reductions with bounded adaptivity* (see Section 5.3 for formal definitions of these restricted reduction notions). We begin in Section 5.1 with a generalization of the notion of PCPs to "randomized" PCPs (which we call AM-PCPs), which can then be used to construct standard PCPs. We also give a generalization of Theorem 4.2. We then introduce bounded-adversary primitives in Section 5.2. The generalized theorem and bounded-adversary primitives are then used together in Section 5.3 to obtain improved PCPs.

5.1. Generalized PCPs and Theorem 4.2.

AM-PCPs. Most of our constructions in this section actually yield a slightly relaxed notion of PCPs, which we call AM-PCPs (as their relation to standard PCPs is analogous to the relation between $\mathcal{AM}$ and $\mathcal{NP}$). These are PCPs where the prover and verifier are allowed to share a common random string. Completeness and soundness are required to hold w.h.p. over the (uniformly random) choice of this string. These are of independent interest (corresponding to hardness of approximation under randomized Karp reductions, see below), and in many cases they can later be converted into standard PCPs (sometimes under assumptions).

Definition 5.1 (AM-PCP $(\mathcal{P}, \mathcal{V})$). *An* AM-PCP *for a language $\mathcal{L} \in \mathcal{NTIME}(g(n))$ consists of a non-adaptive (i.e. stateless) machine $\mathcal{P}$ and an oracle machine $\mathcal{V}$, which both share a random string $z \in \{0,1\}^{b(n)}$. $\mathcal{P}(x, w)$ gets an n-bit input x, advice $w \in \{0,1\}^{g(|x|)}$ (usually a witness to the input's membership in $\mathcal{L}$), the random string z and an input oracle query. $\mathcal{V}(x)$ gets the input x and random string z. The requirements are:*

- *Completeness $(c(n), \gamma_c(n))$. For every $x \in \mathcal{L}$ and corresponding witness w for x's membership in $\mathcal{L}$, with probability at least $1 - \gamma_c(n)$ over z, $\mathcal{V}(x, z)^{\mathcal{P}(x,w,z)}$ accepts with probability at least $c(n)$ (over $\mathcal{V}$'s coins).*
- *Soundness $(s(n), \gamma_s(n))$. For every n-bit input $x \notin \mathcal{L}$ and every non-adaptive cheating $\mathcal{P}^*$ oracle, with probability at least $1 - \gamma_s(n)$ over z, $\mathcal{V}(x, z)^{\mathcal{P}^*(x,z)}$ accepts with probability at most $s(n)$ (over $\mathcal{V}$'s coins).*

Some remarks are in order. First, throughout this work we will be concerned with the case where $\gamma_s(n) = 0$ (i.e. soundness holds for *every* random string). Also, as mentioned above AM-PCPs, beyond being a natural notion in their own right, correspond to hardness of approximation under randomized Karp reductions. An AM-PCP can be converted into a standard PCP (with completeness $c - \gamma_c$ and soundness $s + \gamma_s$) by having the verifier choose z and include it in its oracle queries. This increases the PCP length by a $2^{b(n)}$ multiplicative factor. Moreover, there are several ways of reducing the amount of shared randomness used by an AM-PCP ($b(n)$), and getting a shorter standard PCP. See the remark below.

Remark 5.2. *When $\gamma_s = 0$ and both the verifier and honest prover are efficient (e.g. computable in time* $\mathrm{poly}(n)$*), AM-PCPs can be de-randomized under standard complexity assumptions. This shortens the shared random string's length. To do this, generate the shared random string using a pseudo-random generator, e.g. that of Impagliazzo and Wigderson* [30]*. Under appropriate complexity assumptions, this yields $b(n) = O(\log n)$. See Proposition 5.10 and its proof for the main ideas (used in a slightly different context). If $\gamma_s = 0$ and the honest prover is* inefficient, *AM-PCPs can be de-randomized using the generators of* [35] *or* [37]*. This again gives $b(n) = O(\log n)$, under the (worst-case) assumption that $\mathcal{EXP} = \mathcal{DTIME}(2^{O(n)})$ requires exponential-sized nondeterministic circuits.*

We note that this derandomization is different than (and simpler than) Zimand's approach to derandomizing PCPs [45] *due to the fact that we start with $\gamma_s = 0$. In contrast, soundness is the difficulty for Zimand's approach, and thus it only yields a PCP that is sound against cheating provers with limited access to the shared randomness. Our situation is similar to that of* [7]*, who derandomize certain cryptographic protocols using pseudorandom generators that fool algorithms of fixed polynomial (nondeterministic) circuit size, even though the final protocols need to be secure against arbitrary polynomial-sized adversaries. Like here, their derandomization works because the generators are only needed to preserve properties that depend on honest parties, which run in a fixed polynomial time.*

As a final remark, another approach to reducing or eliminating the shared randomness is via non-uniformity. This goes along the lines of the proof of Proposition 5.9 below. By the probabilistic method, there are $t = O(n/\gamma_c)$ fixed strings $z_1, \ldots, z_t$ such that for every $x \in \mathcal{L} \cap \{0,1\}^n$ completeness holds for at least a $1 - 2\gamma_c$ fraction of the z_i's. Thus, hardwiring $z_1, \ldots, z_t$ as nonuniform advice, only $\log t = \log n + \log(1/\gamma_c) + O(1)$ shared random bits are needed to select a random $i \leftarrow \{1, \ldots, t\}$.

Probabilistic candidate generator. To get unconditional results and results under weaker (worst-case) assumptions, we need to generalize Property 4.1. We need to extend that property to the case where we do not have a deterministic generator that outputs a single hard implementation, but rather a probabilistic generator outputs a hard implementation (for a specific algorithm) w.h.p.

Property 5.3. *The reduction $\mathcal{R}$ (with soundness $s(n)$) has a* probabilistic *polynomial-time generation procedure $\mathcal{G}(1^n)$ that outputs a candidate C in $\mathcal{C}$ such that for every $x \in \{0,1\}^n$ and advice string $w \in \{0,1\}^{\mathrm{poly}(n)}$ given to the prover $\mathcal{P}$:*

$$\Pr_{C \sim \mathcal{G}(1^n)} \left[C \text{ is } \big(\kappa(n), \varepsilon(n)\big)\text{-secure against } \mathcal{S}^{\mathcal{P}(\cdot,x,w)}(\cdot, x) \right] \geq 1 - \gamma(n)$$

where $\gamma(n)$ is a parameter. The randomness complexity *$b = b(n)$ of $\mathcal{G}$ is the number of random bits it uses.*

Generalized Theorem 4.2 Note that now, when we want to use the generic transformation of Theorem 4.2 for a reduction with a probabilistic generator a la Property 5.3, we need for the PCP proof to depend on the hard candidate C. To do this, we use an AM-PCP, where the shared random string determines C. We modify $(\mathcal{P}_{\mathrm{PCP}}, \mathcal{V}_{\mathrm{PCP}})$ accordingly. This is formalized as a generalization of Theorem 4.2. Alternatively, we could then transform the AM-PCP into a standard PCP (increasing the PCP length by a $2^{b(n)}$ multiplicative factor).

Theorem 5.4. *Let $\mathcal{R} = (\mathcal{P}, \mathcal{V}, \mathcal{S}, (\mathcal{C}, \mathcal{T}), \kappa(\cdot), \varepsilon(\cdot))$ be a reduction from an argument system for a language $\mathcal{L}$ to a cryptographic primitive specified by $(\mathcal{C}, \mathcal{T})$, as in Definition 3.2. Suppose furthermore that $\mathcal{R}$ satisfies Property 5.3 and has a probabilistic generator $\mathcal{G}$ for hard candidates that has randomness complexity $b(n)$ and with parameter $\gamma(\cdot)$ such that for all n, $\gamma(n) \leq 1/4$.*

Let $c = c(n)$ and $s = s(n)$ be the completeness and soundness of the argument system, and take $\varepsilon = \varepsilon(n)$, $\kappa = \kappa(n)$, $b = b(n)$, and $\gamma = \gamma(n)$. Recall that $v = v(n)$ bounds the communication from $\mathcal{V}$ to $\mathcal{P}$, the value $u = u(n)$ bounds $\mathcal{P}$'s answer lengths, the value $r = r(n)$ bounds the number of rounds, and $q = q(n)$ bounds the number of $\mathcal{P}^*$-queries made by $\mathcal{T}^{\mathcal{S}^{\mathcal{P}^*}}(C, 1^n, 1^{\lceil 1/\varepsilon \rceil})$.

Then $(\mathcal{P}_{\mathrm{PCP}}, \mathcal{V}_{\mathrm{PCP}})$ is an AM-PCP for $\mathcal{L}$ with completeness $(c - \alpha, \gamma)$ and soundness $(\max\{s, \alpha\}, 0)$. The number of queries is $r + O(\log(1/\alpha) \cdot q)$. The alphabet size is 2^u. The length of the PCP is 2^v. Furthermore, the PCP oracle can be evaluated in time polynomial in the running time of $\mathcal{P}(C, x, w)$ and $\mathcal{G}(1^n)$. The running time of the PCP verifier is polynomial in the running time of $\mathcal{G}(1^n)$, of $\mathcal{V}(C, x)$ and of $\mathcal{T}^{\mathcal{S}^{\mathcal{P}^*}}(C, 1^n, 1^{\lceil 1/\varepsilon \rceil})$.

Proof. The proof is identical to that of Theorem 4.2, except that we use the shared randomness as the coin tosses of $\mathcal{G}$ to generate the candidate C. The analysis is the same, except that completeness only holds when the candidate C is secure against $\mathcal{S}^{\mathcal{P}(\cdot,x,w)}(\cdot,x)$, which holds with probability at least $1-\gamma(n)$ over the shared randomness. Thus, we obtain an AM-PCP with completeness $(c-\alpha,\gamma)$. □

5.2. Bounded-adversary PRFs and CRHFs. In Section 4.2 we obtained PCPs based on cryptographic assumptions. As noted previously, however, the type of hardness we need is much more relaxed than what is usual in the cryptographic setting: we only need hardness for a *specific* algorithm $\mathcal{S}^{\mathcal{P}}$. In this setting, for algorithms that access C as a black box, we can even obtain unconditional results. For example, to an algorithm that makes only q oracle queries, a q-wise independent hash function "looks like" a truly random function. We can use this intuition to transform (black-box) constructions of arguments from collision-resistant hash families (CRHFs) or one-way functions into AM-PCPs (and PCPs) unconditionally or under relatively mild complexity assumptions. The price we pay beyond the (conditional) results of Section 4.2, is that the AM-PCP randomness, and with it the PCP length and verifier running time, may become large.

We define and later construct bounded-adversary pseudorandom function families (PRFs) and collision-resistant hash families (CRHFs). These are function families that (from black-box access) look random or collision resistant (respectively) to a bounded adversary. We do not bound the (polynomial) time needed to compute the function: it may be a larger polynomial than the adversary's running time (note that we do still require that the function is computable in polynomial time). We will then show that bounded-adversary PRF families (*i*) suffice for building a candidate generator instantiating the generic construction of Theorem 5.4. That is, they can be used to build AM-PCPs from black-box reductions (see Section 5.3) from several different cryptographic primitives to argument systems, and (*ii*) can be constructed unconditionally or under weak worst-case complexity assumptions (with various seed lengths). We will also define bounded-adversary collision-resistant hash families (a weaker primitive), show that they can be used to construct a PCP from any black-box reduction from an argument system to CRHF families, and present efficient constructions.

Definition 5.5 (Pseudorandom function). *Consider an ensemble* $\mathcal{F} = \{f_n : \{0,1\}^{j(n)} \times \{0,1\}^{k(n)} \to \{0,1\}^{\ell(n)}\}_n$ *with seed length* $j(n)$*, input length* $k(n)$ *and output length* $\ell(n)$*. We say that* $\mathcal{F}$ *is a* $(s(\cdot),\varepsilon(\cdot))$-pseudorandom function (PRF) *family if for every (nonuniform) circuit-size* $s(n)$ *adversary* $\mathcal{A}$ *(an oracle* *circuit ensemble), for all but finitely many n's:*

$$\left| \Pr_{\text{seed} \sim_R \{0,1\}^{j(n)}} \left[\mathcal{A}^{f_n(\text{seed},\cdot)}(1^n) = 1\right] - \Pr_{\textit{random function } r : \{0,1\}^{k(n)} \to \{0,1\}^{\ell(n)}} \left[\mathcal{A}^r(1^n) = 1\right] \right| \leq \varepsilon(n)$$

I.e. no circuit-size s adversary can distinguish a random function in the family from a truly random function (except with advantage ε).

Sometimes it is easier to construct function families that do not look pseudorandom, but are collision-resistant. In particular, this will be the case for adversaries of bounded adaptivity. When restricting the adversary's adaptivity, we consider the adversary as an oracle circuit with black-box access to the CRH. If the adversary has adaptivity a then the depth of the oracle calls on any path from the circuit's output to an input is at most a. We proceed with a definition of bounded adaptivity oracle algorithms, and then bounded-adversary collision-resistant hash functions.

Definition 5.6 ($a(\cdot)$-Bounded adaptivity algorithm). *A circuit ensemble $\mathcal{A} = \{\mathcal{A}_n^{f_n}\}_n$ with oracle access to a function (ensemble) f has bounded adaptivity $a(\cdot)$ if:*

(i) Each oracle call of $\mathcal{A} = \mathcal{A}_n$ to $f = f_n$ is associated with an adaptiveness level $i \in \{0, \dots, a(n)\}$. We require that each query made by $\mathcal{A}$ is actually of the form (i, y) where i is the adaptiveness level, and y is the input to f.[8]

(ii) For every input $x \in \{0,1\}^n$, on every path from $\mathcal{A}$'s output to an input, the adaptiveness levels of the oracle queries are strictly decreasing.

As discussed above, we define CRHFs against adversaries with bounded adaptivity (and circuit size). These adversaries can only access the CRHF as a black-box, and are further restricted in the total adaptivity of their oracle accesses (and also in terms of their total circuit size).

[8] More generally it would suffice to be able to efficiently extract the adaptiveness level from any query.

Definition 5.7 (Collision-resistant functions). *Consider an efficiently constructible ensemble $\mathcal{F} = \{f_n : \{0,1\}^{j(n)} \times \{0,1\}^{k(n)} \to \{0,1\}^{\ell(n)}\}_n$ with seed length $j(n)$, input length $k(n)$ and output length $\ell(n)$, where $k(n) > \ell(n)$. We say that $\mathcal{F}$ is a $(s(\cdot), a(\cdot), \varepsilon(\cdot))$-*collision resistant function (CRHF) *family if no (nonuniform) circuit-size $s(n)$ adversary $\mathcal{A}$ (an oracle circuit ensemble) with adaptivity $a(n)$ (as in Definition 5.6) can find a collision on a random function from the ensemble with probability ε or greater.*

Note here that we do not bound the complexity of *computing* the pseudorandom and collision-resistant functions, and in particular the function might not be computable by size s circuits (or circuits with depth a). This is similar to complexity-theoretic pseudorandom generators such as those of Nisan and Wigderson [39]. We outline several constructions of bounded-adversary pseudorandom and collision resistant functions. The first is an unconditional construction of PRFs that uses a large seed. The second construction replaces the large seed with a nonuniform construction that uses only a short seed. Then we show how to shorten the seed without resorting to nonuniformity by derandomizing the unconditional construction, using the pseudorandom generators of [39, 30]. The final construction is an unconditional construction of CRHFs for bounded-depth adversaries. The seed length will also be significantly reduced. We begin with the unconditional construction:

Proposition 5.8. *For any input and output lengths $k(n)$ and $\ell(n)$, there exists an $(s(n), 0)$-pseudorandom function. The seed length is $j(n) = s(n) \cdot \max(k(n), \ell(n))$. The function can be evaluated in (uniform) time* $\mathrm{poly}(s(n), k(n), \ell(n))$.

Proof. The pseudorandom function family is a collection of $s(n)$-wise independent functions from $\{0,1\}^{k(n)}$ to $\{0,1\}^{\ell(n)}$ (cf., [1]). A random function in this family looks perfectly random to any algorithm that makes at most $s(n)$ queries. This is because even given the function's value on any up to (adaptively chosen) $s(n) - 1$ points, its value on any other point is completely random (over the choice of the function from the family). Such an $s(n)$-wise independent function can be generated using a seed of $s(n) \cdot \max(k(n), \ell(n))$ random bits and in time $\mathrm{poly}(s(n), k(n), \ell(n))$. □

The main disadvantage of this construction is the large seed length (as large as the adversary's circuit size). We can reduce this seed length by using nonuniformity:

Proposition 5.9. *For any input and output lengths* $k(n), \ell(n)$ *and circuit size* $s(n)$*, there exists a* $(s(n), 1/s(n))$*-pseudorandom function. The seed length is* $j(n) = O(\log s(n))$*. The function can be evaluated in* nonuniform *time* $\mathrm{poly}(s(n), k(n), \ell(n))$*. In fact, a random advice string of this length will yield a PRF of these parameters with probability at least* $1 - 2^{-s(n)}$*. Hence, this can be viewed as a construction in the Common Random String (CRS) Model.*

Proof. We construct the PRF by choosing uniformly at random a set B of $\mathrm{poly}(s(n))$ seeds for the PRF of Proposition 5.8. For any fixed circuit-size $s(n)$ distinguisher D, with probability at least $1-2^{-\mathrm{poly}(s(n))}$ over this choice of B, the distinguisher D's behavior on a random seed from the set B is $1/s(n)$-close to its behavior given a truly random seed. Taking a union bound, with probability at least $1 - 2^{-s(n)}$ over the choice of B no circuit-size $s(n)$ distinguisher has a $1/s(n)$ advantage in distinguishing a random seed from B from a truly random seed. In conclusion, the PRF whose seed is of length $\log |B| = O(\log s(n))$ and is used to choose a (larger) seed from B and compute that larger seed's function, is a $(s(n), 1/s(n)$-PRF. □

Another way of reducing the seed length without resorting to nonuniformity is derandomizing. We can use derandomization techniques, e.g. the work of Impagliazzo and Wigderson [30], to reduce the seed length without hurting pseudorandomness too much. The idea here is to use a pseudorandom generator that stretches a short seed into a longer PRF seed that is indistinguishable from a uniformly random seed to the a fixed circuit-size distinguisher. Here the running time of the generator is allowed to be larger than the (fixed) running time of the distinguisher. To build such a pseudorandom generator, we must make mild (worst-case) complexity assumptions. This is the approach taken in Proposition 5.10.

Proposition 5.10. *Assume that there is a function in DTIME*$(2^{O(n)})$ *that has circuit complexity* $2^{\Omega(n)}$*. Then for any input and output lengths* $k(n), \ell(n)$ *and circuit size* $s(n)$*, there exists a* $(s(n), 1/s(n))$*-pseudorandom function. The seed length is* $j(n) = O(\log(s(n) \cdot \max(k(n), \ell(n))))$*. The function can be evaluated in (uniform) time* $\mathrm{poly}(s(n), k(n), \ell(n))$*.*

Proof. If there is a function in $\mathcal{DTIME}(2^{O(n)})$ that has circuit complexity $2^{\Omega(n)}$, then by the results of [39, 30] there is a pseudorandom generator that stretches $O(\log t(n))$ bits to $t(n)$ bits s.t. no distinguisher of circuit-size $t(n)$ has advantage greater than $1/t(n)$ in distinguishing the generator's output on a random seed from a truly random string. We take $t(n) = \max\{2s(n) \cdot$

$\max(k(n), \ell(n)), s(n) \cdot q(n)\}$, where $q(n) = \mathrm{poly}(s(n), k(n), \ell(n))$ is the time to evaluate the PRF of Proposition 5.8.

Consider then the pseudorandom function that gets a seed of length $O(\log t(n))$ for the generator, stretches it to a string of length $t(n)$, and uses the first $2s(n) \cdot \max(k(n), \ell(n))$ bits of that string as a seed for the *"large seed"* pseudorandom function of Proposition 5.8. Any algorithm that distinguishes the "large seed PRF" when it is evaluated using this pseudorandom seed from the "large seed PRF" function evaluated using a truly random seed can be used (together with the "large seed PRF" evaluator) to break the pseudorandom generator with the same advantage. A (circuit-) size $s(n)$ "large seed PRF" adversary thus corresponds to a size $s(n) \cdot q(n)$ adversary for the PRG, and this means that no size $s(n)$ PRF adversary has advantage greater than $1/t(n)$ in distinguishing the "large seed PRF" using a pseudorandom seed from the "large seed PRF" using a truly random seed. In particular, since with a truly random ("large") PRF seed the distinguishing advantage for a size $s(n)$ adversary from a random function is 0, with the pseudorandom seed the distinguishing advantage of a size $s(n)$ adversary between the PRF and a truly random function is at most $1/t(n) < 1/s(n)$. □

Remark 5.11. *In Proposition 5.10 we made a relatively strong complexity assumption, i.e. we assumed that there is a function in $\mathcal{EXP} = \mathcal{DTIME}(2^{O(n)})$ that has circuit complexity $2^{\Omega(n)}$. In general, we could use more relaxed assumptions to obtain a weaker de-randomization and longer seed length, via* [42]. *For clarity, we focus only on the "high-end" assumption made above. For each setting of parameters, these assumptions about the circuit complexity of $\mathcal{EXP}$) seem to be weaker than assuming the existence of cryptographic pseudorandom functions or collision resistant hash functions with the kind of seed length and security we wish to obtain. The reason is that a cryptographic function with seed length κ (as well as input and output length κ) yields a function in $\mathcal{DTIME}(2^{O(\kappa)})$ whose hardness is related to the security of function.*

Another approach for reducing the seed length is considering bounded adaptivity adversaries (a la Definition 5.6), and settling for collision-resistance (rather than pseudorandomness). Here the idea is that a non-adaptive adversary cannot (information theoretically) find collisions in a pairwise-independent function (except with small probability). For an adversary with fixed adaptivity, we can construct a CRHF by picking the function computed at each level of adaptivity from a pairwise-independent hash family. This will ensure that the adversary cannot (except with small probability) find any collisions, and it is the approach taken below.

Proposition 5.12. *For any input and output lengths $k(n), \ell(n)$, circuit size $s(n)$, and adaptiveness bound $a(n)$, there exists an $(s(n), a(n), a(n) \cdot s^2(n)/2^{\ell(n)})$ -collision resistant function. The seed length is $j(n) = 2(a(n)+1) \cdot \max(k(n), \ell(n))$. The function can be evaluated in (uniform) time* $\mathrm{poly}(a(n), k(n), \ell(n))$.

Proof. Take $a = a(n), k = k(n), \ell = \ell(n)$. Recall that for an adversary $\mathcal{A}$ with bounded adaptivity, each query made by the adversary has an adaptivity level between 0 and $a(n)$, and this adaptivity level is given to the hash function as part of its input. Here we choose a hash function h that includes $a+1$ pairwise-independent hash functions $h_0, \ldots, h_a : \{0,1\}^k \to \{0,1\}^{\ell - \log a}$. The hash function h on input (i, y) (i.e. adaptivity level i and data y) outputs $(i, h_i(y))$. That is, we use a separate hash function for each level of adaptivity. We now claim that the probability that $\mathcal{A}$ finds a collision is at most $a \cdot s^2/2^\ell$.

To bound the probability that $\mathcal{A}$ finds a collision, first recall that for a pairwise independent hash function $h_i : \{0,1\}^k \to \{0,1\}^{\ell - \log a}$, the probability of a collision in any set of q non-adaptive (i.e. fixed before the choice of h) queries is less than $a \cdot q^2/2^\ell$. For each pair of queries the collision probability is $a/2^\ell = 1/2^{\ell - \log a}$, there are less than q^2 pairs of queries. For the hash function we chose, collisions occur only within the same adaptivity level, i.e. on two queries of the form (i, y) and (i, y'). When $\mathcal{S}$ runs, in each level of adaptivity it is making non-adaptive queries to a new hash function. For any fixing of the hash functions up to level $i-1$ and the randomness of $\mathcal{S}^{\mathcal{P}}$, the probability of a collision in level i is thus at most $a \cdot q_i^2/2^\ell$ (over the choice of h_i), where q_i is the number of queries made in level i. Fixing the number of queries in adaptivity level i to be q_i, the total probability of a collision is (taking a union bound over the levels of adaptivity) at most $\sum_{i=0}^{a} a \cdot q_i^2/2^\ell$. Since we know that $\sum_{i=0}^{a} q_i \leq s$, this is at most $a \cdot s^2/2^\ell$. The number of queries made in level i, q_i, is itself a random variable, and so we get that for any adversary $\mathcal{A}$:

$$\begin{aligned}
&\Pr_{\mathcal{A}\text{'s coins}, h_0, \ldots, h_a} [\text{collision in some level}] \\
&\leq E_{\mathcal{A}\text{'s coins}} \left[\sum_{i=0}^{a} E_{h_0, \ldots, h_{i-1}} \left[\Pr_{h_i} [\text{collision in level } i] \right] \right] \\
&\leq E_{\mathcal{A}\text{'s coins}} \left[\sum_{i=0}^{a} E_{h_0, \ldots, h_{i-1}} \left[a \cdot q_i^2/2^\ell \right] \right] \\
&= (a/2^\ell) \cdot E_{\mathcal{A}\text{'s coins}, h_0, \ldots, h_a} \left[\sum_{i=0}^{a} q_i^2 \right] \\
&\leq a \cdot s^2/2^\ell .
\end{aligned}$$

Where the last inequality is because it is always the case that $\sum_{i=0}^{a} q_i \leq s$. The description size of each h_i is less than $2 \cdot \max(k, \ell)$ (see [14]), so the description of h is of size at most $2(a(n) + 1) \cdot \max(k, \ell)$. The function can be evaluated in (uniform) time $\text{poly}(a(n), k(n), \ell(n))$. □

5.3. Instantiations using bounded-adversary primitives. In this section we show that restricted reductions, black-box reductions and reductions with bounded adaptivity, combined with the bounded-adversary primitives of Section 5.2, yield AM-PCPs with improved parameters. We begin with formal definitions of restricted reductions.

Definition 5.13 (Black-box reduction.). *A reduction $\mathcal{R} = (\mathcal{P}, \mathcal{V}, \mathcal{S}, (\mathcal{C}, \mathcal{T}), \kappa(\cdot), \varepsilon(\cdot))$ is a* (fully) black-box *reduction if it is a reduction, as in Definition 3.2, and also $\mathcal{P}$ and $\mathcal{S}$ only have black-box access to C, i.e. they access C as an oracle. Here $t(n)$ also bounds the number of oracle calls made by $\mathcal{S}^{\mathcal{P}}$ (as $t(n)$ is the total combined size of this procedure).*

Definition 5.14 (Black-box reduction with $a(\cdot)$ adaptivity.). *A reduction $\mathcal{R} = (\mathcal{P}, \mathcal{V}, \mathcal{S}, (\mathcal{C}, \mathcal{T}), \kappa(\cdot), \varepsilon(\cdot))$ is a black-box reduction* with adaptivity $a(n)$ *if it is a black-box reduction as in Definition 5.13, and for every candidate C in $\mathcal{C}$ also $\mathcal{S}^{\mathcal{P}(C,x,w)}$ (run with the* honest *prover), is an $a(\cdot)$-bounded adaptiveness algorithm in its oracle calls to C (see Definition 5.6).*

A good example to keep in mind when thinking about bounded-adaptivity reductions is a hash-tree constructed using collision-resistant hashing, say a hash function that shrinks its input by a factor of 2. Here a long string is divided into blocks, each pair of blocks is hashed, the hashes are divided into pairs and each pair is itself hashed, then each pair of hashes of hashes is hashed etc., until a single "hash root" is obtained. The number of hash levels is logarithmic in the number of blocks. This hash-tree construction is a primary component of the argument systems of [33, 36, 6], and both the constructions and the security reductions in these argument systems have logarithmic adaptivity in the sense of Definition 5.14 (see Appendix A for further discussion of these constructions).

Bounded-adversary PRF families can be used to transform reductions from argument systems to one-way functions, PRF families or CRHF families into PCPs. This is done by showing that for any such reduction, the bounded-adversary PRF family can be used to obtain a generator $\mathcal{G}$ satisfying Property 5.3. The fixed and bounded adversary for this PRF is the reduction run with the honest prover: $\mathcal{S}^{\mathcal{P}(\cdot,x,w)}(\cdot, x)$. Recall that this procedure is of circuit-size at most $t(n)$ (and in particular, it makes at most $t(n)$ oracle queries to the PRF family).

For reductions to one-way functions (say functions from $\{0,1\}^\kappa$ to $\{0,1\}^\kappa$), the generator simply outputs a bounded-adversary PRF chosen at random from the family with input and output length κ. Since $\mathcal{S}^{\mathcal{P}(\cdot,x,w)}(\cdot,x)$ cannot distinguish (from its bounded oracle access) this function from a random one, w.h.p. it cannot invert the function on a random input.

The situation is similar for reductions to CRHF families (say with input length 2κ and output and seed length κ). The generator generates a bounded-adversary PRF chosen at random from the family with input length 2κ and output length κ. This function $g(\cdot)$ is interpreted as a CRHF $f(x,y) = g(x)$ by ignoring the first argument (the seed to a function in the family). Since $\mathcal{S}^{\mathcal{P}(\cdot,x,w)}(\cdot,x)$ cannot distinguish (from its bounded oracle access) the function $g(\cdot)$ from a random one, w.h.p. it cannot find collisions on the family $f(\cdot,\cdot)$.

Reductions from an argument system to PRF families (say with input length 2κ and output and seed length κ) are handled in a slightly different manner. Here we cannot just ignore the seed and have a family of size 1 (a random function from any PRF family of size 1 is easily distinguished from a random function). Instead, we use a bounded-adversary PRF family $\{f_s : \{0,1\}^\kappa \times \{0,1\}^{2\kappa} \to \{0,1\}^\kappa\}$. The generator $\mathcal{G}$ outputs a random member f_s of the bounded-PRF family by choosing a random seed s. We interpret this as a PRF family by parsing the first input argument $t \in \{0,1\}^\kappa$ as the index to a function in the PRF, and the second argument as the actual input. Now since $\mathcal{S}^{\mathcal{P}(\cdot,x,w)}(\cdot,x)$ cannot distinguish $f_s(\cdot,\cdot)$ from a truly random function (from its bounded oracle access), it should not be able to distinguish $f_s(t,\cdot)$ from a truly random function. This gives us a generator for reductions to PRF families.

These three cases are captured in the three claims below. The claims and proofs for one-way functions and CRHF families are similar and appear together.

Claim 5.15. *Let $\mathcal{R} = (\mathcal{P}, \mathcal{V}, \mathcal{S}, (\mathcal{C}, \mathcal{T}), \kappa(\cdot), \varepsilon(\cdot))$ be a black-box reduction from a computationally sound argument system to a one-way function $f_n : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ (or from a CRHF family $f_n : \{0,1\}^\kappa \times \{0,1\}^{2\kappa} \to \{0,1\}^\kappa$), where $\kappa = \kappa(n)$. Let $\gamma(\cdot)$ be a parameter, and let $t(n)$ be an upper bound on the running time of $\mathcal{S}^{\mathcal{P}(x,w)}(x)$. Suppose there exists a $(\lambda \cdot t(n), \delta)$-PRF as in Definition 5.5 that can be evaluated in time* poly(n) *and has seed length $j(n)$, input length $\kappa(n)$ (or $2\kappa(n)$ for the CRHF case) and output length $\kappa(n)$, where $\delta = 1/2 \cdot (\gamma \cdot \varepsilon - t(n)^2/2^\kappa)$ and $\lambda > 0$ is a fixed universal constant. Then $\mathcal{R}$ satisfies Property 5.3, with the given $\gamma(n)$ and where the number of coins used by the generator $\mathcal{G}$ is $j(n)$.*

Proof of Claim 5.15. The proof is presented for the case of CRHF families, the case of one-way functions is similar. The generator $\mathcal{G}$ outputs the $j(n)$-bit seed for a bounded-adversary PRF $\mathcal{F}$ as above. This PRF is interpreted as a CRHF by ignoring the first input and taking the second argument to be the actual input. We claim that with probability 1-γ over the choice of seed to $\mathcal{F}$ this is a (κ, ε)-secure CRHF against $\mathcal{S}^{\mathcal{P}}$. Otherwise, if $\mathcal{S}^{\mathcal{P}}$ can with probability γ (over the choice of the $j(n)$-bit seed) find collisions on this CRHF with probability ε, then there is a distinguisher that distinguishes the PRF $\mathcal{F}$ from a truly random function with advantage $\gamma \cdot \varepsilon - t(n)^2/2^{\kappa} > \delta$, a contradiction.

To see this, we construct a PRF-distinguisher for $\mathcal{F}$ (with black-box access to a random PRF or a truly random function). The distinguisher gets oracle access to a function (in $\mathcal{F}$ or truly random). It chooses a random index, runs $\mathcal{S}^{\mathcal{P}}$ on the CRHF formed by fixing the first κ input bits of its oracle function to that index, and outputs 1 if $\mathcal{S}^{\mathcal{P}}$ finds a collision. The size of this distinguisher is at most $\lambda \cdot t(n)$. When run on a random PRF in $\mathcal{F}$, the probability that the distinguisher outputs 1 is at least $\gamma \cdot \varepsilon$. When run on a truly random function, the CRHF we get is a family of random functions, and so the probability that $\mathcal{S}^{\mathcal{P}}$ finds a collision and the distinguisher outputs 1 is at most $t(n)^2/2^{\kappa}$. □

Claim 5.16. *Let $\mathcal{R} = (\mathcal{P}, \mathcal{V}, \mathcal{S}, (\mathcal{C}, \mathcal{T}), \kappa(\cdot), \varepsilon(\cdot))$ be a black-box reduction from a computationally sound argument system to a one-way function $f_n : \{0,1\}^{\kappa} \to \{0,1\}^{\kappa}$ or to a CRHF (or PRF) $f_n : \{0,1\}^{\kappa} \times \{0,1\}^{2\kappa} \to \{0,1\}^{\kappa}$, where $\kappa = \kappa(n)$. Let $\gamma(\cdot)$ be a parameter, and let $t(n)$ be an upper bound on the running time of $\mathcal{S}^{\mathcal{P}(x,w)}(x)$. Suppose there exists a $(\lambda \cdot t(n) \cdot \log(1/\gamma) \cdot 1/\varepsilon^2, \gamma/2)$-PRF as in Definition 5.5 that can be evaluated in time* poly(n) *and has seed length $j(n)$, input length $3\kappa(n)$ and output length $\kappa(n)$, where $\lambda > 0$ is a fixed universal constant. Then $\mathcal{R}$ satisfies Property 5.3, with the given $\gamma(n)$ and where the number of coins used by the generator $\mathcal{G}$ is $j(n)$.*

Proof. The construction and proof for PRF families is similar to the proof of Claim 5.15. The generator $\mathcal{G}$ outputs the $j(n)$-bit seed for a bounded-adversary PRF $\mathcal{F}$ as above. This PRF is interpreted as a PRF family by parsing the first input argument as the index to a function in the PRF family, and the second argument as the actual input.

If $\mathcal{S}^{\mathcal{P}}$ has with probability γ advantage ε in distinguishing the "small PRF" (meaning the PRF built by choosing an index and fixing the first κ bits of the original function in $\mathcal{F}$ to that index) from a "small" (i.e., from $\{0,1\}^{2\kappa}$ to $\{0,1\}^{\kappa}$) truly random function, then we build a distinguisher for $\mathcal{F}$ with total advantage $\gamma/2$. This $\mathcal{F}$-distinguisher gets black-box access to a PRF in $\mathcal{F}$ or random function. It then executes $O(\log(1/\gamma)\cdot 1/\varepsilon^2)$ runs of $\mathcal{S}^{\mathcal{P}}$ on "small functions" produced by fixing the first input bits of its oracle function to random distinct indices, and also $O(\log(1/\gamma)\cdot 1/\varepsilon^2)$ executions of $\mathcal{S}^{\mathcal{P}}$ on truly random "small functions". If the gap between the fraction of executions that output 1 using the input oracle function and truly random functions is at least $\varepsilon/2$, then the distinguisher outputs 1. Now we know that if this distinguisher is given a random function in $\mathcal{F}$, then with probability γ the gap between the probability of 1 in the two executions is at least ε. Thus (by a Chernoff bound) the distinguisher will output 1 with probability at least $3\gamma/4$. If the distinguisher is given a truly random function, then the probability of 1 in both sets of executions is identical, and (by a Chernoff bound) the distinguisher outputs 1 with probability less than $\gamma/4$. The circuit size of this new distinguisher is bounded by $\lambda\cdot t(n)\cdot\log(1/\gamma)\cdot 1/\varepsilon^2$. □

Similarly, for reductions from an argument system to CRHFs with bounded adaptivity (say with input length 2κ and output length κ), we use a bounded-depth-adversary CRHF family $\{f_s : \{0,1\}^{2\kappa}\to\{0,1\}^{\kappa}\}$ (the "original" family). The generator $\mathcal{G}$ outputs a random function from bounded-CRHF family. We interpret this as a family of CRHFs (the "derived" family) by ignoring the first input argument, and taking the second argument as the actual input (similarly to the construction of Claim 5.15). Now since $\mathcal{S}^{\mathcal{P}(\cdot,x,w)}(\cdot,x)$ cannot find *any* collision on f_s chosen at random from the "original" family (from its bounded oracle access), it should not be able to find f_s-collisions in the "derived" family (where the seed is ignored). This gives us a candidate generator for reductions from CRHFs.

CLAIM 5.17. *Let $\mathcal{R} = (\mathcal{P},\mathcal{V},\mathcal{S},(\mathcal{C},\mathcal{T}),\kappa(\cdot),\varepsilon(\cdot))$ be a black-box reduction with adaptivity $a(n)$ from a computationally sound argument system to a CRHF $f_n : \{0,1\}^{2\kappa}\to\{0,1\}^{\kappa}$, let $\gamma(\cdot)$ be a parameter, and let $t(n)$ be an upper bound on the running time of $\mathcal{S}^{\mathcal{P}(x,w)}(x)$. Let $\mathcal{F}$ be a $(\lambda\cdot t(n), a, \delta)$-CRHF family as in Definition 5.5 that is computable in time* poly(n) *and has seed length $j(n)$, input length $2\kappa(n)$ and output length $\kappa(n)$, where $\delta = 1/2\cdot(\gamma\cdot\varepsilon - t(n)^2/2^{\kappa})$ and $\lambda > 0$ is a fixed universal constant. Then $\mathcal{R}$ satisfies Property 5.3, with the given $\gamma(\cdot)$ and where the number of coins used by the generator $\mathcal{G}$ is $j(n)$.*

PROOF. The proof is similar to that of Claim 5.15. The generator $\mathcal{G}$ outputs the $j(n)$-bit seed for a bounded-adversary CRHF $f\in\mathcal{F}$ as above. This "original" CRHF f is interpreted as a family of "derived" CRHFs by ignoring the first input argument and taking the second argument as the actual input.

With probability 1-γ over the choice of seed to $\mathcal{F}$ this is a (κ, ε)-secure CRHF against $\mathcal{S}^{\mathcal{P}}$. Otherwise, if $\mathcal{S}^{\mathcal{P}}$ can with probability γ (over the choice of the $j(n)$-bit seed) find collisions on the randomly chosen "derived" CRHF family (which ignores its seed) with probability ε, then $\mathcal{S}^{\mathcal{P}}$ can be used to break the "original" family $\mathcal{F}$ with probability $\gamma \cdot \varepsilon$. The $\mathcal{F}$-adversary gets a random function $f(\cdot)$ from $\mathcal{F}$, and runs $\mathcal{S}^{\mathcal{P}}$ on the "derived" family $f(x, y) = g(y)$, to find a collision. With probability $\gamma \cdot \varepsilon$, this returns a collision in the "original" family $\mathcal{F}$. The circuit size of this $\mathcal{F}$-adversary is about the same as the circuit size of $\mathcal{S}^{\mathcal{P}}$ (up to say some constant multiplicative factor). When run on a truly random function, the CRHF we get is a random function, and so the probability that $\mathcal{S}^{\mathcal{P}}$ finds a collision and the distinguisher outputs 1 is at most $t(n)^2/2^{\kappa}$. □

Instantiating the generic transformation. Recall from Sections 5.1, 5.2 the generic transformation of Theorem 5.4 and also the parameters it gives on known reductions. We instantiate this transformation, converting reductions from pseudorandom functions or collision-resistant hash families into AM-PCPs (or standard PCPs), using the bounded-adversary PRF families and CRHF families of the previous section. Note that in this setting it even makes sense to consider reductions with logarithmic security parameter (logarithmic in the running time of the bounded adversary).

Unconditional PRF. Any reduction from arguments to one-way functions, PRF families or CRHF families yields (unconditionally) an AM-PCP using the bounded-adversary PRF families of Proposition 5.8 together with Claims 5.15 and 5.16. The completeness, soundness, query complexity and alphabet size are as in the theorem statement of Theorem 5.4. The main "price" of this instantiation is the amount of shared randomness used by the AM-PCP that is obtained. The number of bits needed to choose a function in the family is $O(t \cdot \kappa)$, where t is the circuit size of $\mathcal{S}^{\mathcal{P}}$ (e.g. poly(n) for arguments with efficient provers). If we wanted to translate this AM-PCP into a standard PCP, the length of the PCP would thus become exponential: $2^{v+O(t\cdot\kappa)}$. While this length is large, constructing even such exponential length PCPs from scratch (e.g. the Hadamard PCP of [2]) is quite nontrivial. Another disadvantage is that the verifier's running time becomes fairly large (polynomial in t).

The shared randomness used by the AM-PCP (and thus the length of the standard PCP from it) can be improved either using nonuniformity or derandomization. These techniques can be applied either to the final AM-PCP (following Remark 5.2), or directly to the PRFs, as we describe now:

Nonuniform unconditional PRF. Continuing the discussion above, if we use the nonuniform bounded-adversary PRF of Proposition 5.9, the number of bits needed to choose a function in the family becomes only $O(\log t)$. The length of the standard PCP we can obtain shrinks to $2^v \cdot \mathrm{poly}(t)$. The verifier's running time, however, remains polynomial in t, and moreover the prover and verifier are now nonuniform. An alternative approach that avoids the nonuniformity (at the cost of making complexity assumptions) is derandomization.

Derandomized conditional PRF. The (almost) final approach we suggest for transforming reductions into PCPs is shortening the seed length of PRF families using derandomization under (worst-case) complexity assumptions. If we assume that for some $\beta > 0$, it holds that there is a function in $DTIME(2^{O(n)})$ that does not have circuits of size $2^{\beta \cdot n}$, then we can use the PRF of Proposition 5.10. The number of bits needed to choose a function in the family becomes only $O(\log t + \log(1/\varepsilon))$.[9] As before, the length of the standard PCP we can obtain shrinks to $2^v \cdot \mathrm{poly}(t, 1/\varepsilon)$. The verifier's running time, while uniform, still remains polynomial in t.

Remark 5.18. *If we are willing to make stronger assumptions, we can assume here the existence of one-way permutations* $f : \{0,1\}^\kappa \to \{0,1\}^\kappa$ *that are hard to invert for circuits of size* $2^{\Omega(\kappa)}$. *This would give (using the works of* [11, 43, 24, 23]*), a* $(s, 1/s)$*-PRF families with seeds of length* $O(\log s)$ *that can be computed in (uniform) time* $\mathrm{poly}(\log s(n))$ *and give efficient verifier running time and a standard PCP with short length.*

Finally, an alternative that lets us reduce the PCP length and verifier running time is considering bounded-adaptivity black-box reductions.

Bounded-adaptivity CRHF. If we have a reduction to CRHFs with bounded adaptivity a, we can use the (unconditional) bounded-adversary CRHF of Proposition 5.12 together with Claim 5.17. The number of bits needed to choose a random function in the family is $O(a(n) \cdot \kappa)$. The verifier's running time becomes polynomial in that of the PCP verifier and in $a(n)$ and κ. The length of the standard PCP we can obtain shrinks to $2^v \cdot 2^{O(a(n)\cdot\kappa)}$.

To get an idea of what these results mean for the known reductions, observe that in known reductions we can choose κ to be logarithmic, and for a CRHF reduction with logarithmically bounded adaptivity (e.g. that of [33]), we get

[9] Note that we could make milder assumptions about the hardness of $DTIME(2^{O(n)})$ and obtain weaker derandomizations (i.e. longer seed) as in Remark 5.11.

(unconditionally) a (standard) PCP of only quasipolynomial length. Using such reductions to (unconditionally) construct a PCP of *polynomial* length remains an open question.

Acknowledgements

Preliminary versions of this paper appeared in *24th IEEE Conference on Computational Complexity* and on the *Electronic Colloquium on Computational Complexity* [41].

We thank Oded Goldreich for illuminating conversations and encouragement, Luca Trevisan for an old discussion which led to the bounded-adversary pseudorandom functions we use in Section 5, and the anonymous CCC 2009 and computational complexity reviewers for their helpful comments.

Guy Rothblum's research was done mostly while at MIT, U.C. Berkeley and Microsoft Research, and supported by NSF Grants CCF-0635297, NSF-0729011, CNS-0430336 and CCF-0832797 and by a Computing Innovations Fellowship.

Salil Vadhan's work was done in part while visiting U.C. Berkeley, supported by the Miller Institute for Basic Research in Science and a Guggenheim Fellowship, and was also supported by NSF grant CNS-0831289.

References

[1] N. Alon & J. H. Spencer. *The Probabilistic Method*. Wiley, New York, 1992.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan & M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[3] S. Arora & S. Safra. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122, 1998.

[4] L. Babai, L. Fortnow, L. A. Levin & M. Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31, 1991.

[5] B. Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115, 2001.

[6] B. Barak & O. Goldreich. Universal arguments and their applications. *SIAM J. Comput.*, 38(5):1661–1694, 2008.

[7] B. Barak, S. J. Ong & S. P. Vadhan. Derandomization in cryptography. *SIAM J. Comput.*, 37(2):380–400, 2007.

[8] M. Ben-Or, S. Goldwasser, J. Kilian & A. Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *STOC*, pages 113–131, 1988.

[9] M. Ben-Or, S. Goldwasser & A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[10] E. Ben-Sasson & M. Sudan. Short pcps with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008.

[11] M. Blum & S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.

[12] G. Brassard, D. Chaum & C. Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[13] R. Canetti, O. Goldreich & S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[14] L. Carter & M. N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[15] D. Chaum, C. Crépeau & I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[16] I. Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, pages 445–456, 1991.

[17] I. Dinur. The pcp theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

[18] U. Feige, S. Goldwasser, L. Lovász, S. Safra & M. Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

[19] L. Fortnow, J. Rompel & M. Sipser. On the power of multi-prover interactive protocols. *Theor. Comput. Sci.*, 134(2):545–557, 1994.

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

[21] O. Goldreich. *The Foundations of Cryptography – Volume 2*. Cambridge University Press, 2004.

[22] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[23] O. Goldreich, S. Goldwasser & S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[24] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *STOC*, pages 25–32, 1989.

[25] O. Goldreich, S. Micali & A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *FOCS*, pages 174–187, 1986.

[26] O. Goldreich, S. Micali & A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[27] O. Goldreich, S. P. Vadhan & A. Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 11(1-2):1–53, 2002.

[28] J. Groth. Linear algebra with sub-linear zero-knowledge arguments. In *CRYPTO*, pages 192–208, 2009.

[29] I. Haitner & T. Holenstein. On the (im)possibility of key dependent encryption. In *TCC*, pages 202–219, 2009.

[30] R. Impagliazzo & A. Wigderson. = *BPP* if requires exponential circuits: Derandomizing the xor lemma. In *STOC*, pages 220–229, 1997.

[31] Y. Ishai, E. Kushilevitz & R. Ostrovsky. Efficient arguments without short pcps. In *IEEE Conference on Computational Complexity*, pages 278–291, 2007.

[32] Y. T. Kalai & R. Raz. Probabilistically checkable arguments. In *CRYPTO*, pages 143–159, 2009.

[33] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

[34] J. Kilian. Improved efficient arguments (preliminary version). In *CRYPTO*, pages 311–324, 1995.

[35] A. Klivans & D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002.

[36] S. Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453, 1994.

[37] P. Bro Miltersen & N. V. Vinodchandran. Derandomizing arthur-merlin games using hitting sets. *Computational Complexity*, 14(3):256–279, 2005.

[38] M. Naor. On cryptographic assumptions and challenges. In *CRYPTO*, pages 96–109, 2003.

[39] N. Nisan & A. Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

[40] O. Reingold, L. Trevisan & S. P. Vadhan. Notions of reducibility between cryptographic primitives. In *TCC*, pages 1–20, 2004.

[41] G. N. Rothblum & S. P. Vadhan. Are pcps inherent in efficient arguments? In *IEEE Conference on Computational Complexity*, pages 81–92, 2009.

[42] C. Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003.

[43] A. C.-C. Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91, 1982.

[44] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[45] M. Zimand. Probabilistically checkable proofs the easy way. In *IFIP TCS*, pages 337–351, 2002.

A. Known argument constructions

In this section we briefly review known constructions of argument systems from cryptographic primitives, and their parameters in terms of our notions of reduction. The approaches taken in the works of Kilian [33], Micali [36] and Barak and Goldreich [6] are all based on collision-resistant hash functions (or random oracles) and polynomial-length PCPs (polynomial in the time needed to recognize the language). The work of Ishai, Kushilevitz and Ostrovsky [31] is based on homomorphically additive encryption over a large (super-polynomial) field and exponential length PCPs (exponential in the time needed to recognize the language), where each symbol in the PCP is a linear function of its location in the PCP (the locations are viewed as vectors over a large field). We review the constructions from CRH and homomorphic encryption (we do not review the random-oracle construction of [36]).

Overview of constructions. In all of these constructions, the cryptographic primitive is used by the prover to "commit" to a (long) PCP proof string. The (short) computationally binding commitment can be sent to the verifier. This commitment allows de-committing to individual bits (or symbols) of the PCP very efficiently (in terms of the verifier's work and the communication). To build an argument system, the prover commits to a PCP and sends the commitment to the argument verifier. This verifier runs the PCP verifier who requests some bits of the PCP, the argument verifier requests these bits and their decommitment from the (argument) prover. The argument prover sends the bits and decommitments, the verifier verifies that they are valid, completes the simulation of the PCP verifier and accepts if it accepts.

To argue computational soundness, the intuition is that unless the prover can break the commitment scheme, once it sends the commitment it is essentially bound to a single PCP string, and the PCP's soundness (against non-adaptive provers) carries over to the argument setting.

To be more precise, consider the distribution D_i of (non-adaptive) queries made by the PCP verifier, chosen uniformly and at random conditioned on the PCP verifier querying location i. The distribution D_i (given i) is efficiently sampleable for the PCPs used. Consider the following decommitment experiment using any (potentially cheating) prover: run the commitment scheme on a PCP string of the prover's choice, then choose a random verifier query set and a random query i from that set, take two samples from D_i, and run the prover (twice) to decommit to the each of the two index sets. Compare the two answers to query i. The commitment schemes have the property that if in this experiment with probability at least ε the prover de-commits to two different values as the i-th symbol, then this prover can be used to generate an adversary $\mathcal{A}$ that breaks the underlying cryptographic primitive (collision resistant hash or homomorphic encryption) with advantage $\varepsilon^{\Omega(1)}$ (in the sense of Definition 3.1).[10] The number of queries $\mathcal{A}$ makes to the cheating prover is constant, i.e. $O(1)$.

Soundness of the argument systems follows from the soundness s_{PCP} of the PCP and ε-security of the primitive. Let $q = O(1)$ be the PCP's query complexity. If the primitive is ε secure, then by the above, the prover's cheating probability in the decommitment experiment is at most ε. From this we conclude that the argument's soundness is at least $s_{\text{PCP}} - (\varepsilon \cdot q)^{O(1)}$: roughly, this because with high probability over the coins of the commitment phase, with high probability over the verifier's query, there is only a single high-probability answer that the prover gives to this query. Changing the prover to *always* return this high-probability answer makes it non-adaptive without changing the system's behavior much. This means that the adaptive but computationally bounded prover could not have cheated with too high probability to begin with (since an unbounded but non-adaptive prover cannot break the PCP's soundness).

The parameters. Turning our attention to the parameters of these constructions, in the constructions of [33, 36, 6], collision-resistant hash functions that shrink their input by a multiplicative factor of 2, say from $\{0,1\}^{2\kappa}$ to $\{0,1\}^{\kappa}$, are used to build a hash tree of logarithmic depth (logarithmic in the PCP length). The commitment is the root of the hash tree. To decommit to a symbol in the PCP the prover reveals the (logarithmically many) values along the path from the root to the requested symbol. The completeness c and soundness are (more or less) inherited from the PCP, the parameter ε is polynomial in the soundness and query complexity of the original PCP, as is the number of queries q made by $\mathcal{T}^{\mathcal{S}^{\mathcal{P}^*}}$ to a cheating prover oracle. The circuit size t of $\mathcal{T}^{\mathcal{S}^{\mathcal{P}}}$ is polynomial in the PCP length and the soundness. The communication complexity is logarithmic in the PCP length, and polynomial in the number of queries and the security parameter κ. The number of rounds of communication is $O(1)$. Note also that the hash tree being used can very naturally be framed in terms of a *bounded-adaptivity* reduction, where the number of levels of adaptivity is logarithmic in the PCP length.

[10] For the [31] reduction, we are assuming that the field size here is larger than some polynomial in $1/\varepsilon$.

In summary, if we consider a PCP for an $\mathcal{NP}$ language with small constant soundness and completeness, and constant query complexity we get an argument with (slightly worse) constant soundness and completeness. The security proof uses a cheating prover to break the CRH with small constant advantage (polynomial in the soundness). The number of queries made by $\mathcal{T}^{\mathcal{S}^{\mathcal{P}^*}}$ to a cheating prover is constant, and its circuit size when run with the honest prover is polynomial. The number of rounds is constant and the communication complexity is $\log n \cdot \kappa$. Viewed as a bounded-adaptivity reduction, the adaptivity is logarithmic.

Turning to the reduction of [31], which uses homomorphic encryption and using a linear PCP, the parameters are similar, except that the running times of the honest prover and of $\mathcal{T}^{\mathcal{S}^{\mathcal{P}}}$ depend logarithmically on the PCP length. We also note that the communication from the prover to the verifier is proportional only to κ (not even logarithmic in the PCP length).

Other related work. Kilian [34] gave constructions of arguments from collision-resistant hash families with improved efficiency, this improved construction still falls into our framework (and indeed uses PCPs). Groth [28] gives efficient arguments for specific algebraic functionalities and also for circuit satisfiability (based on specific assumptions). These constructions achieve communication that roughly proportional to the square-root of the circuit size. When plugged into our reductions, such parameters do not imply a non-trivial PCP. Kalai and Raz [32] construct computationally sound PCPs (Probabilistically Checkable Arguments PCAs) using some PCP techniques and a computational Private Information Retrieval protocol. This construction also falls into our framework. Their PCAs are short (polynomial only in the witness length), but require that the verifier first send a message (or public key) to the prover. This message causes the PCP we could obtain from the reduction to be significantly longer.

Number of queries to the reduction. As a final note, observe that in our results the number of queries made by the PCP verifier grows with the number of queries the reduction proof of security makes to the cheating prover. In known constructions of argument systems, i.e. those outlined above, the number of queries to the reduction is a fixed polynomial in the cheating prover's success probability. This is the case in many (but certainly not all) reductions in cryptography. In hybrid arguments, for example, (a standard proof technique for cryptographic reductions) polynomially many queries are sometimes necessary. For reductions from computationally sound argument systems, if the number of queries was larger - e.g. polynomial, we would not obtain an interesting PCP. This remains a promising avenue for future research (as discussed in the introduction).

Manuscript received 26 September 2009

Guy N. Rothblum
Computer Science Department
Center for Computational Intractability
Princeton University
Princeton, NJ 08540, USA
rothblum@csail.mit.edu

Salil Vadhan
School of Engineering & Applied Sciences
and
Center for Research on Computation & Society
Harvard University
Cambridge, MA 02138, USA
salil@seas.harvard.edu