Electronic Colloquium on Computational Complexity, Report No. 45 (2012)

On the Concrete-Efficiency Threshold of Probabilistically-Checkable Proofs*

Eli Ben-Sasson† eli@cs.technion.ac.il Technion
Alessandro Chiesa† alexch@csail.mit.edu MIT CSAIL
Daniel Genkin† danielg3@cs.technion.ac.il Technion
Eran Tromer‡ tromer@csail.mit.edu Tel Aviv University

April 22, 2012

Abstract

Probabilistically-Checkable Proofs (PCPs) form the algorithmic core that enables succinct verification of long proofs/computations in many cryptographic constructions, such as succinct arguments and proof-carrying data.

Despite the wonderful asymptotic savings they bring, PCPs are also the infamous computational bottleneck preventing these cryptographic constructions from being used in practice. This reflects a lack of understanding of how to study and improve the concrete (as opposed to asymptotic) efficiency of PCPs.

We propose a new efficiency measure for PCPs (and its major component: locally-testable codes and PCPs of proximity). We define a *concrete-efficiency threshold* that indicates the smallest problem size beyond which the PCP becomes "useful", in the sense that using it actually pays off relative to naive verification by simply rerunning the computation; our definition takes into account *both* the prover's and verifier's complexity.

We then show that there exists a PCP with a *finite* concrete-efficiency threshold. This does not follow from existing works on PCPs with succinct verifiers. We provide a PCP construction where the prover and verifier are efficient enough to achieve a finite threshold, and further show that this PCP has the requisite properties for being used in the cryptographic applications mentioned above.

Moreover, we extend and improve the Reed-Solomon PCPs of proximity over fields of characteristic 2, which constitute the main component in the quasilinear-size construction of [Ben-Sasson and Sudan, STOC '05] as well as our construction. Our improvements reduce the concrete-efficiency threshold for testing proximity to Reed-Solomon codes from 2^{572} in their work to 2^{35}, which is tantalizingly close to practicality. We hope this will motivate the search for even better constructions, ultimately leading to practical PCPs for succinct verification of long computations.

Keywords: PCPs; low-degree tests; Reed-Solomon code; verification of computation

*We thank Ohad Barta and Arnon Yogev for reviewing prior versions of this paper.

†The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 240258.

‡This work was supported by the Check Point Institute for Information Security, by the Israeli Ministry of Science and Technology, and by the Israeli Centers of Research Excellence I-CORE program (center 4/11).

ISSN 1433-8092

Contents

1 Introduction

The study of *probabilistically-checkable proofs* (PCPs) was initiated by Babai et al. [BFLS91] and Feige et al. [FGL+96] with two very different motivations. Babai et al. focused on "positive" applications of PCPs: enabling *succinct* verification of long computations. On the other hand, Feige et al. focused on "negative" applications of PCPs: proving inapproximability results.

In this paper we initiate the study of *practical PCPs*, by which we mean the study of the *concrete* (as opposed to *asymptotic*) efficiency of PCPs, via cost functions that are informed by the efficiency concerns that arise in *positive* applications of PCPs. We present three contributions: the first is a definitional one, the remaining two are of a technical nature.

We introduce a definition for the *concrete-efficiency threshold* B of a given PCP system $\mathcal{S}$. Informally, B is the smallest value T for which using $\mathcal{S}$ to verify computations of running time at least T does in fact offer savings over naive verification (i.e., in which the prover writes down all T steps of the computation and the verifier reads this transcript from start to end).

Our first result is that there exists a PCP system with finite concrete-efficiency threshold. As we shall argue, this is not the case for existing PCP constructions. We further show that our PCP construction has the requisite properties for use in cryptographic constructions like computationally-sound proofs of knowledge [Mic00, Val08], proof-carrying data [CT10], succinct non-interactive arguments of knowledge (SNARK) [BCCT11], and a slightly weaker (though still meaningful) variant of universal arguments [BG02].

As an intermediate step to computing the concrete-efficiency threshold of our PCP system (which is quite challenging), we study instead the concrete-efficiency threshold of the "heaviest" component in our construction: PCPs of proximity (PCPPs) for Reed-Solomon codes over fields of characteristic 2. These, as in [BSS08], lie at the heart of our PCP system. Using an improved analysis of these PCPPs we obtain our second result: a reduction in the concrete-efficiency threshold of PCPPs for Reed-Solomon codes from 2^{572} in [BSS08] to 2^{35}.

The ultimate goal of our work is to make the vision of Babai et al. a reality via practical PCP implementations. The quest for practical PCPs reveals a mostly-uncharted landscape, whose exploration is an intriguing research direction that we believe will lead to interesting insights and techniques, with the potential of great impact in light of the many useful cryptographic constructions that crucially rely on PCPs.

1.1 Motivation

PCPs are the "verification engine" of many cryptographic protocols that enable various flavors of succinct verification of long computations, including *succinct arguments* [Kil92, Mic00, BG02], non-interactive succinct arguments [DCL08, BCCT11, DFH11, GLR11], and even *distributed* succinct arguments [CT10]. Unfortunately, PCPs are also the notorious computational bottleneck of all these cryptographic protocols: despite the great practical potential of succinct verification of long computations, any construction relying on PCPs is dismissed as impractical.

Theoretical results have already established the feasibility of succinct probabilistic verification of long computations [BFLS91, BSGH+05, Mie09]: it is possible to encode a computation with only a polynomial (in fact, even quasilinear) blowup in length and then verify it in polylogarithmic time. However, these results are only *asymptotic* results. In fact, a lack of explicit constructions, tight complexity analyses, and proof details currently make it difficult to even assess how far current PCP constructions are from practical efficiency.

We believe that this situation should change: we should strive to understand the *concrete* (rather than asymptotic) efficiency of PCPs and see how much such efficiency can be "pushed" in an effort to realize the unfulfilled impact of one of the most beautiful results theoretical computer science can claim.

1.2 Goals

Given a program $\mathbb{P}$ (say, written in `C++`), an input x, and a time bound T, a verifier wishes to check whether a witness w causes $\mathbb{P}(x, w)$ to accept within T time steps. Without the help of a prover, the verifier can only perform the naive verification procedure: run $\mathbb{P}(x, w)$ for at most T steps and check if it halts and accepts within that time.

As [BFLS91] showed, a PCP saves the verifier the cost of performing this naive verification procedure, and enables instead a much faster *succinct* probabilistic verification procedure: a PCP verifier can run in only $\mathsf{poly}(|\mathbb{P}|, |x|, \log T)$ time when having oracle access to a PCP proof string π of length $\mathsf{poly}(|\mathbb{P}|, |x|, T)$ generated by a PCP prover (who is also given w as auxiliary input). More recent works [BSGH+05, Mie09] achieved a PCP proof length $T \cdot \mathsf{poly}(|\mathbb{P}|, |x|, \log T)$. As T grows larger (and viewing $|x|$ as relatively small) the saving in running-time becomes more dramatic, and at some point running the PCP verifier is much better than naive verification.

Two things complicate the matter. The first is that the known PCP constructions with succinct verifiers have huge overheads, indicating that real savings are made only for prohibitively large T. The second complication is that the PCP prover must convert the computation-transcript (which is of length T) into a PCP proof for the PCP verifier. This cost is at least as large as the length of a PCP proof but — if not constructed by efficient algorithms — can be much larger, so much so as to kill any prospect of using a PCP system in practical settings.

Therefore, the crucial parameters of interest from the point of view of using PCPs for succinct verification of long computations are the *prover running time* and the *verifier running time*.[1] While clearly these times can be traded off against each other (e.g., increasing proof length, and thus prover running time, enables the design of more "robust" tests with faster verifiers), we want them instead to *simultaneously* be as short as possible. This sets the ground for the basic question studied in this paper:

> **Our Question:**
> Can we design a definition that captures the *concrete efficiency* of a given PCP system? Do there exist PCP systems with good concrete efficiency?

Such a definition could provide a *new efficiency measure* for the study of PCPs, one that is different than other traditional efficiency measures (such as query and randomness complexity) and better captures what we look for in a PCP that is to be used in positive applications, by taking into account both the prover running time and verifier running time.

1.3 Summary of Our Results

(R1) Defining the concrete efficiency of a PCP system. We suggest to capture the concrete efficiency of a given PCP system by defining a *concrete-efficiency threshold* B that indicates the problem size beyond which naive verification of a computation actually takes longer than

[1] The running time of the verifier depends on the target soundness. Our convention is to consider a target soundness of $1/2$; then, k-fold sequential repetition results in soundness 2^{-k} and a multiplicative blowup in running time of k. Also see Remark 6.2.

using the PCP system; in other words, B is the smallest problem size beyond which the PCP system always guarantees that using it is better than not using it.

To instantiate the above approach into a definition, we first need to agree on a meaningful *cost function* $\mathbf{C}$. As already discussed in Section 1.2, the two parameters of interest are the prover running time (or, equivalently, proving overhead) and verifier running time. So we will let the cost function $\mathbf{C}$ take these two as input in order to compute a combined cost, and then compare this cost against the cost of naive verification. More precisely, the concrete-efficiency threshold of a PCP system is the smallest problem size beyond which the (combined) cost output by $\mathbf{C}$ is always smaller than a *savings factor* γ times the cost of naive verification, for a given $\gamma \in (0, 1)$.

Definition 1 (informal)**.** *A PCP system is* B**-efficient** *with respect to cost function* $\mathbf{C}$ *and savings factor* γ *if* B *is the smallest integer for which every true statement* y *of size at least* B*,*

$$\mathbf{C}\begin{pmatrix}\textit{PCP proving}\\ \textit{overhead for } y\end{pmatrix}, \begin{pmatrix}\textit{PCP verification}\\ \textit{time for } y\end{pmatrix} < \gamma \cdot \begin{pmatrix}\textit{naive verification}\\ \textit{time for } y\end{pmatrix} . \tag{1}$$

Depending on the application, different choices of cost function $\mathbf{C}$ and savings factor γ may be meaningful. In this paper, we choose $\mathbf{C}(a, b) \stackrel{\text{def}}{=} a \cdot b$ and $\gamma \stackrel{\text{def}}{=} 1/2$ as a natural convention.

For concreteness, we need to choose a computational model (through which provers and verifiers will be represented) as well as a "reference" complete problem relative to which a given PCP is constructed. We choose circuits for both: our definition assumes that provers and verifiers are specified as Boolean circuits, and considers PCP systems constructed for the NEXP-complete language Succinct Circuit Sat (or SCS for short), which is the language of satisfiable circuits that are *succinctly* represented.[2] These choices are not arbitrary: analogous alternative definitions for other choices are of course possible, but one must be careful.

Indeed, in moving from an asymptotic study of efficiency to a concrete one, we must be very explicit about how we model computation. For example, fixing an *arbitrary* universal Turing machine as a model computation will make the definition of B meaningless: it could be that the universal Turing machine we fixed has all problems of size less than 2^{1000} precomputed, and would cause $B = 1$ for any PCP system that becomes useful just before 2^{1000}. To avoid such problems, we would need to specify a "natural" universal Turing machine *explicitly*, but then we would have a tough choice about which one to pick. By choosing circuits, instead, we can be explicit about the model of computation without much work (e.g., by specifying a natural Boolean basis and settings for fan-in and fan-out).

Henceforth our goal is to construct B-efficient PCP systems for SCS with as small a B as possible.

(R2) A PCP system with $B < \infty$. It is not clear that a PCP system with a finite efficiency threshold even exists, and such a system does not follow from previous works on the topic. Indeed, PCP constructions designed for inapproximability results (and most constructions fall under this category) have both prover and verifier running in time that is at least linear in T, and therefore have an infinite efficiency threshold of ∞ (i.e., none exists) [FGL+96, AS98, ALM+98, PS94, RS97, HS00, GS06, BSSVW03, BSGH+04, BSS08, Mei12, MR08]. Turning to PCP constructions with succinct verifiers [BFLS91, BSGH+05, Mie09], these focus solely on the

[2]Note that it is indeed essential to choose a language that gives the freedom to specify large constraint satisfaction problems succinctly: the verifier should not be forced to read a given constraint satisfaction problem in explicit form because then it would not have a chance to make any savings in running time!

verifier's running time, and none have studied the prover's running time. Careful inspection of these works reveals that the running time can be bounded by a polynomial in T, but if this polynomial is quadratic (or larger) then, once again, the concrete-efficiency threshold is ∞. Another issue is the dependence of the running time of the prover or verifier on the instance size: if it is more than quadratic, then again the concrete-efficiency threshold is ∞.

Our first main theorem states that our definition of concrete efficiency can be fulfilled:

Theorem 1. *There exists a PCP system for* SCS *with* $B < \infty$.

Having established that it is indeed possible to obtain a finite concrete-efficiency threshold, we wish to carefully optimize our PCP construction from the proof of Theorem 1 with the goal of deriving as tight an analysis as possible to show a *small* value of B. Deriving an upper bound on B, however, would entail deriving concrete upper bounds on the size of fairly-complicated circuits implementing the prover and verifier. Such a computation seems infeasible without the aid of a full, working implementation. We propose instead a more tractable, yet still meaningful, approach to study B, which we describe next.

(R3) Towards a small B: concrete efficiency of PCPP verifiers for Reed-Solomon. Our PCP construction from the proof of Theorem 1 is an *algebraic PCP* whose technical heart, like other algebraic PCPs, is a *low-degree test*, where a verifier V is given oracle access to a function f, and possibly also an auxiliary "proximity proof" π, and must test proximity of f to some code of low-degree polynomials by querying (f, π) at a few places.

A low-degree test is a special case of a *PCP of Proximity (PCPP) system* for a code ensemble and, in order to study the concrete efficiency of low-degree tests, we propose a definition (analogous to Definition 1) for the concrete efficiency of PCPP systems for code ensembles. In this new definition, instead of computing a combined cost based on the prover and the verifier running time, we compute a combined cost based on the *length* of the codeword plus its PCPP proof (or, more precisely, blow up in this length relative to the dimension of the code) and the *number of queries* of the verifier.

Definition 2 (informal)**.** *A PCPP system for a given code ensemble $\mathcal{E}$ is $\hat{B}$-***efficient** *if $\hat{B}$ is the smallest integer for all $k \geq B$,*

$$\left(\frac{n(k) + L(k)}{k}\right) \cdot Q(k) < \frac{1}{2} \cdot k$$

where $n(k)$ denotes the block-length, $L(k)$ the length of the PCPP proof generated by the prover, and $Q(k)$ the query complexity of the verifier for codes in $\mathcal{E}$ of dimension k.

The definition above is an "information-theoretic" version of Definition 1 as it disregards the *computational* cost of the prover for producing the proof and of the verifier for generating queries and examining answers, focusing on information-theoretic measures instead (cf. Remark 2.8).

As in [BSS08, BSGH+05], the relevant code ensemble for our PCP construction from the proof of Theorem 1 is the ensemble $\mathcal{E}_{\mathrm{RS}}$ of Reed-Solomon (RS) codes evaluated over subspaces of fields of characteristic 2; let us denote $\hat{B}_{\mathrm{RS}}$ the concrete-efficiency threshold of a given PCPP system for $\mathcal{E}_{\mathrm{RS}}$. The advantage of the simpler Definition 2 is that we will be able to *explicitly* derive upper bounds for the $\hat{B}_{\mathrm{RS}}$ of PCPP systems that we construct. The analysis of the

PCPP system for Reed-Solomon in [BSS08] only shows an efficiency threshold of $\hat{B}_{\mathrm{RS}} \leq 2^{572}$, which is an astronomical upper bound. Our second main theorem significantly improves on it and obtains a PCPP system with small(er) $\hat{B}_{\mathrm{RS}}$:

Theorem 2. *There exists a PCPP system $\mathcal{E}_{\mathrm{RS}}$ with $\hat{B}_{\mathrm{RS}} \leq 2^{35}$.*

Problem sizes on the order of 2^{35} are tantalizingly close to practicality! We are thus very optimistic about the prospect of improving further the concrete-efficiency thresholds of PCPP systems for Reed-Solomon codes and, thus ultimately, of algebraic PCPs.

While we only provide an upper bound on the information-theoretic (and thus weaker) efficiency measure of Definition 2 for the PCPPs we construct, doing so is still meaningful because the algorithmic costs in our PCPP construction (and, ultimately, also in our PCP construction) only account for small additional logarithmic factors relative the PCPP proof length and query complexity; we thus expect B to be not much larger than $\hat{B}_{\mathrm{RS}}$ (cf. Remark 2.10).

(R4) Properties of our PCP system for cryptographic constructions. A PCP system is not used "as is" in positive applications, because one needs to somehow ensure that the verifier has oracle access to a *fixed* PCP string. Instead, the PCP system is plugged into cryptographic constructions that leverage the PCP soundness guarantees — but these constructions often require the PCP system to satisfy additional properties.

Possibly the most important positive application of PCPs is the *construction of succinct arguments* [Kil92, Mic00, BG02], because a practical succinct argument directly implies a *practical delegation scheme for all of* NP — a major open problem in cryptography. In fact, PCPs seem to be inherent to the construction of succinct arguments [RV09] and the efficiency of a succinct argument is usually closely related to the efficiency of the PCP used to construct it. (See Remark 10.3.)

An example of an additional property that is often needed in constructions of succinct arguments is *proof of knowledge*; for instance, it gives the ability to delegate "cryptographic computations" [BCCT11] and to recursively compose non-interactive proofs [CT10, BCCT12]. This property is required when constructing computationally-sound proofs of knowledge [Mic00, Val08], SNARKs [BCCT11], and proof-carrying data [CT10].

Other times even more properties are needed. For example, the construction of universal arguments by Barak and Goldreich [BG02] requires, besides a stronger notion of "implicit" proof of knowledge (and the fact that an efficient prover exists), two other properties: a *non-adaptive verifier* and the existence of an *efficient reverse sampler*. The specific application of universal arguments is noteworthy as (together with the reduction of [Kil92]) it is the only succinct argument whose security reduction relies on standard cryptographic assumptions (in contrast to the security reductions of the succinct arguments in [Mic00, Val08, CT10] and [BCCT11, DFH11, GLR11] that respectively rely on oracle and extractability assumptions).

Thus, when studying positive applications of PCPs, we must not only take into account the efficiency considerations that arise in practice (as discussed in Section 2.1), but also any additional properties that may be needed for actually using PCPs within a given cryptographic application.

Our third theorem states that our PCPs can be used in several cryptographic constructions:

Theorem 3. *The PCPs from the proof of Theorem 1 satisfy the requisite properties for the construction of (a slightly weaker yet still useful variant of) universal arguments [BG02]. In particular, our PCPs also satisfy the requisite properties for the construction of computationally-sound proofs of knowledge [Mic00, Val08], proof-carrying data [CT10], and SNARKs [BCCT11].*

2 Main Results

In this section we formally state our main results, which we discussed at high level in Section 1.3; along the way, we shall give pointers to the relevant technical sections. We start with the definition of the concrete-efficiency threshold of a PCP system (Section 2.1); we then state our first main result about the existence of a PCP system with finite concrete-efficiency threshold (Section 2.2); we then define the concrete-efficiency threshold for PCPPs and state our second main result about PCPPs for Reed-Solomon codes (Section 2.3); finally, we discuss cryptographic properties of the PCPs we construct (Section 2.4).

2.1 Defining the Concrete-Efficiency Threshold of a PCP System

In this section, we formalize our result **(R1)** by defining the concrete-efficiency threshold of a PCP system (which we informally introduced in Definition 1).

For simplicity, we use Boolean circuits with only NAND gates as our model of computation. A *(Boolean) circuit* C is a directed acyclic graph with fan-in and fan-out equal to 2; its size is the number of vertices in this graph. If the graph has n sources and m sinks, then we say that the circuit has n inputs and m outputs. The circuit C then computes a corresponding Boolean function f_C from $\{0,1\}^n$ to $\{0,1\}^m$ in the natural way: the evaluation of an input $x \in \{0,1\}^n$ is performed by placing each bit of x at the corresponding source of the graph (when taking sources, say, in lexicographic order) and then, by considering every non-source non-sink vertex as a NAND gate, a bit for each sink is computed by evaluating the circuit in some topological order; the output $y \in \{0,1\}^m$ is the string of bits at the sinks (when taken, say, in lexicographic order). The circuit C is satisfiable if there is some input x that makes the first bit of the output of $f_C(x)$ equal to 1. We assume a canonical representation of circuits as strings.

We also need to choose a "reference" complete language relative to which to construct PCPs. The language Succinct Circuit Sat (SCS for short) will be quite convenient; informally, SCS is the language of all satisfiable Boolean circuits C over 2^n gates that are *succinctly* represented via a descriptor circuit F having (roughly) n inputs and n outputs. Formally:

Definition 2.1 (Succinct Circuit SAT [Pap94])**.** *Let $n \in \mathbb{N}$, $f\colon \{0,1\}^{n+2} \to \{0,1\}^n$. The circuit corresponding to f, denoted C_f, is the circuit with 2^n gates, each labeled with an n-bit string, such that: for each gate $s \in \{0,1\}^n$, letting $s_{00} = f(s00)$, $s_{01} = f(s01)$, $s_{10} = f(s10)$, and $s_{11} = f(s11)$, gates s_{00} and s_{01} have wires going into s, and gate s has wires going into s_{10} and s_{11}; if $s = s_{00} = s_{01}$ then s is an input gate; if $s = s_{10} = s_{11}$ then s is an output gate. (And if f C_f is not "valid", i.e. a gate receives more than two wires, then C_f is the circuit with no wires.)*

The language Succinct Circuit Sat*, or* SCS *for short, is the language of all circuit descriptors F such that the Boolean function f computed by F is of the form $f\colon \{0,1\}^{n+2} \to \{0,1\}^n$ and C_f is satisfiable. (Note that we can assume without loss of generality that $|F| \leq 2^n$.)*

Informally, a PCP system for SCS is a pair consisting of a prover P and a verifier V that works as follows. The prover receives as input a circuit descriptor F describing a circuit C and

an assignment w for C, and outputs a PCP proof π for the claim "w is a satisfying assignment of C". The verifier receives as input the circuit descriptor F and a string of random bits, and has two outputs: a set of indices (indicating the bits of the proof that are to be read) and a predicate (which decides whether the answers to the queries are satisfactory); we assume that the verifier is non-adaptive, i.e., all queries depend only on the input of the verifier and not on the answers provided by the prover. Because our computational model is circuits, P and V are represented as circuit families, each indexed by two integers n and k where, e.g., $P_{n,k}$ is the circuit that is "specialized" for descriptor circuits F of size k describing circuits of size 2^n.

Thus, analogously to previous definitions of PCP systems (with succinct verifiers) [BFLS91, BSGH+05], we have the following definition:

Definition 2.2 (PCP system for SCS)**.** *Let $P = \{P_{n,k}\}_{n,k\in\mathbb{N}}$ and $V = \{(Q_{n,k}, D_{n,k})\}_{n,k\in\mathbb{N}}$ be circuit families. We call (P, V) a* **PCP system for SCS** *if the following conditions hold:*

- Completeness. *For every (F, w) with $F\colon \{0,1\}^{n+2} \to \{0,1\}^n$ and w is witness to $F \in$ SCS,*

$$\Pr_r\left[D_{n,|F|}(r, F, \pi|_{\vec{q}}) = 1 \;\middle|\; \begin{array}{c} \pi \leftarrow P_{n,|F|}(F, w) \\ \vec{q} \leftarrow Q_{n,|F|}(r, F) \end{array}\right] = 1 \ ,$$

  where $\vec{q}$ is interpreted as a vector of indices and $\pi|_{\vec{q}}$ is the substring of π induced by $\vec{q}$.
- Soundness. *For every $F\colon \{0,1\}^{n+2} \to \{0,1\}^n$ such that $F \notin$ SCS and every $\tilde{\pi}$,*

$$\Pr_r\left[D_{n,|F|}(r, F, \tilde{\pi}|_{\vec{q}}) = 1 \;\middle|\; \vec{q} \leftarrow Q_{n,|F|}(r, F)\right] < \frac{1}{2} \ .$$

The following definition is the main notion motivating our work. Informally, given a PCP system (P, V) for SCS, we define its *concrete-efficiency threshold* B as the smallest n such that, for every circuit F describing a circuit C of size at least 2^n, when using the PCP system (P, V) on input F we save a multiplicative factor $\gamma = 1/2$ over naive verification of C, when taking into account *both* proving overhead and verification time via the cost function $\mathbf{C}(a, b) = a \cdot b$. (Recall that these choices of savings factor and cost function are the convention we fix in this paper.)

Definition 2.3 (concrete-efficiency threshold for PCPs)**.** *Let (P, V) be a PCP system for* SCS *(cf. Definition 2.2). The* **concrete-efficiency threshold** *of (P, V) is the smallest integer B such that for every circuit F computing a Boolean function of the form $f\colon \{0,1\}^{n+2} \to \{0,1\}^n$ with $n \geq B$,*

$$\left(\frac{|P_{n,|F|}|}{|F| \cdot 2^n}\right) \cdot (|Q_{n,|F|}| + |D_{n,|F|}|) < \frac{1}{2} \cdot (|F| \cdot 2^n) \ . \tag{2}$$

If no such integer exists, the concrete-efficiency threshold of (P, V) is infinite (i.e., $B := \infty$).

In the formal definition above we have instantiated the intuitive quantities that appeared in the informal Definition 1: given a circuit descriptor F of size k describing a circuit C of size 2^n (which takes the role of the instance y),

- the naive verification time is given by $k \cdot 2^n$, because C has 2^n gates and computing the input/output wires of each gate of C requires evaluating F on appropriate inputs;
- the prover overhead is given by $\frac{|P_{n,k}|}{k \cdot 2^n}$, because $|P_{n,k}|$ is the size of the appropriate prover divided by $k \cdot 2^n$, which again is the time to understand the topology of C; and
- the verification time is given by $|Q_{n,k}| + |D_{n,k}|$, because $|Q_{n,k}|$ and $|D_{n,k}|$ are the sizes of the appropriate query and decision algorithms.

Remark 2.4. The use of two parameters n and k in Definition 2.3 is necessary: omitting one of them would make Definition 2.3 satisfiable only with $B = \infty$. The is because a circuit of size 2^n can have a descriptor circuit that can itself be as large as 2^n. Similarly, a descriptor circuit of size k can describe a circuit of size between k and 2^k. If only one parameter is used, say k, we would need the verifier to be of size 2^k even though in some cases a circuit described by k bits has size $\mathsf{poly}(k)$. This would imply that no finite concrete-efficiency threshold exists.

2.2 Theorem 1: A B-efficient PCP for SCS with $B < \infty$

In this section, we formalize our result **(R2)** by restating our first main theorem, Theorem 1, more formally; its proof is given in Section 9.

Theorem 1 (restated)**.** *There exists a PCP system for* SCS *(cf. Definition 2.2) with a concrete-efficiency threshold (cf. Definition 2.3) that is finite.*

Note that, in light of Definition 2.3, it is indeed not clear that such a PCP system exists. For example, if $\frac{|P_{n,|F|}|}{|F|\cdot 2^n} = \Omega(2^n)$, or $\frac{|P_{n,|F|}|}{|F|\cdot 2^n} = \Omega(|F|)$, or $(|Q_{n,|F|}| + |D_{n,|F|}|) = \Omega(|F|^2)$, then $B = \infty$. Thus the efficiency requirements imposed on a PCP system so to fulfill Definition 2.3 with *any finite* concrete-efficiency threshold B (not to mention a small B) are already quite stringent.

2.3 Theorem 2: Concrete Efficiency of Proximity Testing to Reed-Solomon

In this section, we formalize our result **(R3)** by defining the concrete-efficiency threshold of a PCPP system for a code ensemble (which we informally introduced in Definition 2) and then restating our second main theorem, Theorem 2, more formally.

Recall that an $[n, k, d]_q$*-linear error correcting code* (or, simply, $[n, k, d]_q$-code) is a k-dimensional subspace C of $\mathbb{F}_q^n$, where $\mathbb{F}_q$ denotes the finite field with q elements and q is a prime-power, such that every pair $w \neq w'$ of members of C differ on at least d positions. (Later on we shall focus on a special ensemble of codes, namely, Reed-Solomon codes over fields of characteristic 2.) In what follows, let $(\mathbb{F}_q^n)^{\leq Q}$ denote the set of vectors in $\mathbb{F}_q^n$ with at most Q nonzero entries.

Definition 2.5 (PCPP for linear code and LTC)**.** *Given $L, Q \in \mathbb{N}$, an (L, Q)-**PCPP system for an** $[n, k, d]_q$-**code** C is a pair (P, V) where P is a linear mapping from $\mathbb{F}_q^k$ to $\mathbb{F}_q^L$ and V is a distribution supported on $(\mathbb{F}_q^n)^{\leq Q}$ satisfying the following conditions:*

- Completeness. *For every codeword $w \in C$ and $\pi := P(w)$ (its associated "proof of proximity"),*

$$\Pr_{v \leftarrow V}\left[\sum_{i=1}^{n+L} v_i \cdot (w \circ \pi)_i = 0\right] = 1 \ ,$$

  where $w \circ \pi$ is the concatenation of w and π and arithmetic operations are carried in $\mathbb{F}_q$.
- Soundness. *For every non-codeword w that is $d/3$-far from C and every $\pi \in \mathbb{F}_q^L$,*

$$\Pr_{v \leftarrow V}\left[\sum_{i=1}^{n+L} v_i \cdot (w \circ \pi)_i = 0\right] < \frac{1}{2} \ .$$

If C has a $(0,Q)$-PCPP we say C is a **Q-query LTC**.

Furthermore, for functions L and Q, an **(L,Q)-PCPP system for a (linear) code ensemble** *$\mathcal{E}$ is a pair (P,V) where, for each $C \in \mathcal{E}$, (P_C, V_C) is a $(L(C), Q(C))$-PCPP system for C. If $L(C) = 0$ for all $C \in \mathcal{E}$, then $\mathcal{E}$ is called an* **ensemble of LTCs** *and V is a* **tester** *for $\mathcal{E}$.*

Remark 2.6 (possible generalizations)**.** The definition above includes a number of simplifying assumptions and arbitrarily fixed constants. We fix the soundness to be $1/2$ for a proximity parameter that is one-third of the minimal distance of the code. The definition is stated only for linear codes. The mapping sending a message to the PCPP for its codeword is assumed to be linear. The verifier is defined to be non-adaptive, its decision predicate is linear, and it has perfect completeness. Although the latter set of assumptions holds without loss of generality for linear LTCs (cf. [BSHR05, Theorem 3.3]), this is not known to hold for general PCPPs, even assuming the code is linear. Since all these assumptions apply to the codes that we study, we prefer a concrete and simple definition to one that holds in greatest generality (which can anyways be easily deduced from Definition 2.5).

We now provide a definition for a concrete-efficiency threshold for PCPPs for linear code ensembles; this definition is "information-theoretic", i.e., it does not impose constraints on the computational resources required by the prover and verifier. This is in stark contrast to Definition 2.3 and we discuss the rationale behind this difference below.

Definition 2.7 (concrete-efficiency threshold for PCPPs)**.** *Let $\mathcal{E}$ be a linear-code ensemble and (P,V) a PCPP system for $\mathcal{E}$ (cf. Definition 2.5). The* **concrete-efficiency threshold** *of (P,V) is the smallest integer $\hat{B}$ such that for every linear code $C \in \mathcal{E}$ that is an $[n,k,d]_q$-code with $k \geq \hat{B}$ it holds that*

$$\left(\frac{n + L(C)}{k}\right) \cdot Q(C) < \frac{1}{2} \cdot k \ . \tag{3}$$

If no such integer exists, the concrete-efficiency threshold of (P,V) is infinite (i.e., $\hat{B} := \infty$).

Remark 2.8 (computational vs. information-theoretic concrete efficiency)**.** The cost in Definition 2.3 takes into consideration the computational complexity of the prover and verifier, whereas the cost in Definition 2.7 above only takes into account proof length and query complexity (disregarding the computational complexity required to produce the proof, to sample a query tuple, and then to verify the query answers).

We transition to an information-theoretic definition for codes so to be able to compute the concrete-efficiency threshold for an important family of codes (cf. Theorem 2). Computing a similar concrete-efficiency threshold by taking computational complexity into account (à la Definition 2.3) would require pinning down concrete upper bounds to the circuit-sizes for finite-field arithmetic and interpolation of polynomials (among other things) — a quite difficult analysis. Nonetheless, studying the information-theoretic definition is still meaningful (cf. Remark 2.10).

Case of interest: additive Reed-Solomon. The quasilinear-size PCPs of [BSS08] are based on PCPPs for the ensemble of Reed-Solomon codes *over additive subgroups of finite fields of characteristic* 2. We define these codes next. Recall that a degree-ℓ extension of the two-element field $\mathbb{F}_2$ is also an ℓ-dimensional linear space over $\mathbb{F}_2$.

Definition 2.9 (additive Reed-Solomon codes over finite fields of characteristic 2)**.** *Given a finite field $\mathbb{F}_q$, a subset S of $\mathbb{F}_q$, and a degree bound d, the* **Reed-Solomon code** $\mathrm{RS}(\mathbb{F}_q, S, d)$

is the $[|S|, d+1, |S|-(d+1)]_q$*-code whose codewords are functions* $f\colon S \to \mathbb{F}_q$ *computed by polynomials of degree at most* d*. The ensemble of* **Reed-Solomon codes over additive subgroups of fields of characteristic** 2*, denoted* $\mathcal{E}_{\mathrm{RS}}$*, is defined to be the ensemble of RS-codes* $\mathrm{RS}(\mathbb{F}_q, S, d)$ *where* q *is a power of* 2 *(i.e.,* $\mathrm{char}(\mathbb{F}_q) = 2$*),* S *is an* $\mathbb{F}_2$*-subspace and* $d = |S|/8$*.*

We can now formally restate Theorem 2, our second main theorem:

Theorem 2 (restated)**.** *There exists a PCPP system (cf. Definition 2.5) for the code ensemble* $\mathcal{E}_{\mathrm{RS}}$ *(cf. Definition 2.9) having a concrete-efficiency threshold* $\hat{B}_{\mathrm{RS}} \leq 2^{35}$ *(cf. Definition 2.7).*

Our strategy for proving Theorem 2 is in two steps: (a) revisit and tighten the Polishchuk–Spielman Bivariate Testing Theorem [PS94] and its corollary that is relevant to us [BSS08, Lemma 6.12]; and (b) provide a class of constructions that generalize the proximity testers of [BSS08] and provide for this class a much tighter and explicit soundness analysis that allows us to numerically evaluate the soundness of the verifier for any given problem size and construction from the class, so that we can then choose a construction with the best $\hat{B}_{\mathrm{RS}}$. The proof of Theorem 2 is given in Section 4.

Remark 2.10. Studying $\hat{B}_{\mathrm{RS}}$ (cf. Definition 2.7) instead of B (cf. Definition 2.3) is justified because:

- The reductions of Ben-Sasson et al. [BSCGT12] are quasilinear (only $O(\log^2 T)$), and thus it suffices to concentrate on studying the concrete efficiency of the algebraic PCP construction.
- A low-degree test is not only the technical heart of an algebraic PCP, but also its efficiency bottleneck. Indeed, while an algebraic PCP necessarily involves other components (such as verifying vanishing properties of functions), these other components are usually "thin layers" on top of a low-degree test; as a rule of thumb, the low-degree test determines most of the concrete efficiency of an algebraic PCP. And this is certainly the case for our algebraic PCP construction (which is given in Section 8). Thus, it suffices to concentrate on the concrete efficiency of proximity testing to, in our case, certain Reed-Solomon codes.
- While $\hat{B}_{\mathrm{RS}}$ does not take into account the running time of the prover, but only its output length, this turns out not to be a problem for the case of Reed-Solomon codes, because the PCPP systems we use only require basic arithmetic in the ring of multivariate polynomials over finite fields of characteristic 2. Indeed, through the use of appropriate additive-FFT techniques [Mat08] and invoking various properties of linearized polynomials [LN97, Section 2.5], we show that the running time of the prover closely follows its output size and the running time of the verifier closely follows its query complexity.[3]

We thus expect the "real" B to not be far from $\hat{B}_{\mathrm{RS}}$.

2.4 Theorem 3: Properties for Cryptographic Constructions

In this section, we discuss result **(R4)** in more detail.

Our Theorem 3 states that the PCPs we construct in the proof of Theorem 1 satisfy properties that are needed in several cryptographic constructions. Specifically, we prove that the PCPs we construct satisfy the following properties:

[3] In fact, we note that it is a non-trivial concern how to make even the verifier run efficiently at all (not to mention in time that is close to the query complexity). Working with univariate low-degree polynomials means that the verifier will have to test proximity for degrees that are enormous; therefore, any "reasoning about high-degree polynomials" must be conducted in a succinct way. The sparsity of linearized polynomials, induced by the use of subspaces as domains of evaluation for the Reed-Solomon code, will enable the verifier to do so.

"Explicit" PCP proof of knowledge. We show that whenever the PCP verifier is convinced with sufficiently large probability, the PCP proof can be decoded to a good witness by running a knowledge extractor that runs in time proportional to the size of the witness. Note that we are not able to construct a knowledge extractor that can locally decode the witness. This lack of local decoding here seems inherent because our PCPs, as those of [BSS08, BSGH+05], rely on "univariate techniques" (cf. Remark 10.39). Hence, because we do not achieve a knowledge extractor that is an implicit representation of the good witness, we call the knowledge property we achieve *explicit* PCP proof of knowledge.

Non-adaptivity of the PCP verifier. We show that the PCP verifier can be "split" into a query algorithm and a decision algorithm. It is rather obvious from our PCP construction that this is the case, but explicitly working out these algorithms for the more complex components of the construction is quite non-trivial.

Efficient reverse-samplability of the PCP verifier. We show that there is an efficient algorithm that, given a query number and an index into the PCP oracle, is able to output a string that is uniformly distributed among those that could indeed have produced the index when given as input the query number — this algorithm is thus an efficient "reverse sampler". That our PCP construction is reverse samplable is not obvious; we show that it is by constructing an efficient reverse sampler for it.[4]

Using our PCPs in cryptographic constructions. The "explicit" PCP proof of knowledge mentioned above falls short of the one needed for the construction of universal arguments. Nonetheless, we show that the same construction as [BG02, Construction 3.4] still gives arguments with a weaker notion of proof of knowledge — these we call *almost universal arguments*. Such a weaker notion is not a problem because it still suffices for most "positive" applications (cf. Remark 10.2).

The properties required of PCPs in the construction of almost universal arguments are a superset of the properties required of PCPs in other cryptographic applications such as [Mic00, Val08, CT10, BCCT11], so it will suffice in the proof to focus on almost universal arguments.

The proof of Theorem 3 is given in Section 10.

[4]This fact was already sketched by [Mie09] for the proximity testers at the base of our PCPs; he used their reverse samplability for showing a verifier-efficient construction of gap amplification [Din07].

3 Definitions

We present formal definitions for the basic notions that we use throughout this paper.

3.1 Reed-Solomon and Reed-Muller Codes

Definition 3.1. *Let $\mathbb{F}$ be a field and S a subset of $\mathbb{F}$. The* **evaluation** *of a polynomial $P(x) = \sum_{i=0}^{d} a_i x^i$ over S is the function $p\colon S \to \mathbb{F}$ defined by $p(s) = P(s)$ for all $s \in S$. The formal sum $P(x)$ is called the polynomial* **corresponding** *to (the function) p. Similar definitions hold for multivariate polynomials.*

Definition 3.2. *Let $\mathbb{F}$ be a field and $d \geq 0$. A polynomial $P(x)$ over $\mathbb{F}$ has* **degree** *d if the degree of P in x is at most d.*

Definition 3.3. *Let $\mathbb{F}$ be a field, S a subset of $\mathbb{F}$, and $d \geq 0$. The* **Reed-Solomon code** *of degree d over $\mathbb{F}$ evaluated at S is the set of functions*

$$\mathrm{RS}(\mathbb{F}, S, d) := \{p\colon S \to \mathbb{F} \text{ s.t. } p \text{ is an evaluation of a polynomial of degree at most } d \text{ over } S\} \ .$$

Definition 3.4. *def:RS Let $\mathbb{F}$ be a field, S and H subsets of $\mathbb{F}$, and $d \geq 0$. The H-***vanishing Reed-Solomon code** *of degree d over $\mathbb{F}$ evaluated at S is the set of functions*

$$\mathrm{VRS}(\mathbb{F}, S, H, d) := \{p \in \mathrm{RS}(\mathbb{F}, S, d) \text{ s.t. the polynomial corresponding to } p \text{ vanishes on } H\} \ .$$

Definition 3.5. *Let $\mathbb{F}$ be a field and $d, e \geq 0$. A polynomial $P(x, y)$ over $\mathbb{F}$ has* **degree** *(d, e) if the degree of P in x is at most d and the degree of P in y is at most e.*

Definition 3.6. *Let $\mathbb{F}$ be a field, W a subset of $\mathbb{F} \times \mathbb{F}$, and $d, e \geq 0$. The* **bivariate Reed-Muller code** *of degree (d, e) over $\mathbb{F}$ evaluated at W is the set of functions*

$$\mathrm{RM}\big(\mathbb{F}, W, (d, e)\big) := \left\{ p\colon W \to \mathbb{F} \text{ s.t. } \begin{array}{l} p \text{ is an evaluation of a bivariate polynomial} \\ \text{of degree at most } (d, e) \text{ over } W \end{array} \right\} \ .$$

3.2 Notions of Distance

Next, we recall the notions of fractional (or relative) distance between strings and to low-degree polynomials. (In general, when we say "close" we shall mean "$\leq$" and when we say "far" we shall mean "$>$".)

The fractional (or *relative*) Hamming distance between two strings is the ratio between the number of positions where the two strings differ divided by their length:

Definition 3.7. *Let Σ be a finite alphabet and N a positive integer. For any two Σ-strings a and b in Σ^N, the* **fractional (Hamming) distance** *over the alphabet Σ between a and b, denoted $\Delta_N(a, b)$, is equal to the number of positions where a and b differ divided by N,*

$$\Delta_N(a, b) := \frac{\big|\{i \in N \,:\, a_i \neq b_i\}\big|}{N} \ .$$

The definition easily extends to functions from N to Σ, as it is possible to think of the Σ-strings a and b as functions $\tilde{a}, \tilde{b}\colon N \to \Sigma$ where $\tilde{a}(i) = a_i$ and $\tilde{b} = b_i$ for each $i \in N$.

The fractional Hamming distance between two functions allows to define the fractional Hamming distance of a function to low-degree polynomials:

Definition 3.8. *Let $\mathbb{F}$ be a field, V a subset of $\mathbb{F}$, $f\colon V \to \mathbb{F}$ a univariate function, and $d \geq 0$. We denote by $\delta_V^{(d)}(f)$ the fractional Hamming distance of f to $\mathrm{RS}(\mathbb{F}, V, d)$, i.e.,*

$$\delta_V^{(d)}(f) := \inf_{\substack{P \in \mathbb{F}[x] \\ \deg_x(P) \leq d}} \Delta_V(f, P) \ .$$

Definition 3.9. *Let $\mathbb{F}$ be a field, W a subset of $\mathbb{F} \times \mathbb{F}$, $f\colon W \to \mathbb{F}$ a bivariate function, and $d, e \geq 0$. We denote by $\delta_W^{(d,e)}(f)$ the fractional Hamming distance of f to $\mathrm{RM}(\mathbb{F}, W, (d, e))$, i.e.,*

$$\delta_W^{(d,e)}(f) = \inf_{\substack{Q \in \mathbb{F}[x,y] \\ \deg_x(Q) \leq d \\ \deg_y(Q) \leq e}} \Delta_W(f, Q) \ .$$

Furthermore, we denote by $\delta_W^{(d,*)}(f)$ the fractional distance when the degree in y is unrestricted, and by $\delta_W^{(*,e)}(f)$ the fractional distance when the degree in x is unrestricted.

Univariate low-degree polynomials form a code with a certain distance:

Lemma 3.10. *Fix a field $\mathbb{F}$, a subset $S \subseteq \mathbb{F}$, and an integer $d \in \mathbb{N}$. For any two distinct polynomials $P, P'\colon \mathbb{F} \to \mathbb{F}$ of degree at most d, it holds that $\Delta_S(P, P') > 1 - \frac{d}{|S|}$.*

In particular, we can deduce the "unique decoding radius" for univariate low-degree polynomials:

Lemma 3.11. *Fix a field $\mathbb{F}$, a subset $S \subseteq \mathbb{F}$, an integer $d \in \mathbb{N}$, and $\delta \in [0, 1]$. If a function $p\colon S \to \mathbb{F}$ is δ-close to $\mathrm{RS}(\mathbb{F}, S, d)$ and $1 - \frac{d}{|S|} > 2\delta$ then there is a* unique *polynomial P of degree at most d whose evaluation over S is δ-close to p.*

Recall that Reed-Solomon codes have an efficient decoding procedure (e.g., via the Welch–Berlekamp algorithm [WB86, GS92]):

Claim 3.12 (Decoding Reed-Solomon Codes)**.** *There exists a polynomial-time algorithm* DecodeRS *that, on input (the representation of) a finite field $\mathbb{F}$, a subset S of $\mathbb{F}$, a degree d (presented in unary), and a function $p\colon S \to \mathbb{F}$, outputs a polynomial $P\colon \mathbb{F} \to \mathbb{F}$ of degree at most d whose evaluation over S is closest to p, provided that p lies in the unique decoding radius.*

3.3 PCPs and PCPPs

We formally introduce *probabilistically-checkable proofs* and *probabilistically-checkable proofs of proximity*, as well as the functions that will help us keep track of their various complexity parameters.

PCPs. We first recall the standard definition of a probabilistically-checkable proof (PCP). For functions $r, q\colon \mathbb{N} \to \mathbb{N}$ we say that a probabilistic oracle machine V is a (r, q)-PCP verifier if, on input a binary string x of length n and given oracle access to a binary string π, V runs in time $2^{O(r(n))}$, tosses $r(n)$ coins, makes $q(n)$ queries to π, and outputs either 1 ("accept") or 0 ("reject").

Definition 3.13. *Let $s \in [0,1]$ and $r, q \colon \mathbb{N} \to \mathbb{N}$. A language L belongs in the class $\mathsf{PCP}_s[r(n), q(n)]$ if there exists a (r,q)-PCP verifier V_L such that the following holds:*

1. **(Perfect) Completeness:** *If $x \in L$ then there exists π such that $\Pr_R[V_L^\pi(x;R) = 1] = 1$.*
2. **Soundness:** *If $x \notin L$ then for every π it holds that $\Pr_R[V_L^\pi(x;R) = 1] \leq s$.*

PCPs of Proximity. A PCP of Proximity (PCPP) [BSGH+06] (also known as an assignment tester [DR04]) is a strengthening of a PCP where the input comes in two parts (x, y), where x, the *explicit input*, is given explicitly to the verifier and y, the *implicit input*, is only given as an oracle to the verifier (and queries to it are counted as part of the query complexity); the explicit input is of the form $x = (x', N)$ and N is the length of y. Because now the verifier may not see the input in its entirety, it only make sense to require verifiers to test theorems for being "close" to true theorems.

More precisely, one considers *pair languages* (which are simply binary relations). For a pair language L and a binary string x, we define L_x to be all the N-bit strings y such that $(x, y) \in L$. Then, for an arbitrary y, we define $\Delta(y, L_x)$ to be 1 if $L_x = \emptyset$ else to be the smallest fractional Hamming distance of y to any element in L_x.

For functions $r, q \colon \mathbb{N} \to \mathbb{N}$ we say that a probabilistic oracle machine V is a (r,q)-PCPP verifier if, given an explicit input $x = (x', N')$ with $|x'| = n$ and oracle access to an N-bit implicit input y and proof π, V runs in time $2^{O(r(n))}$, tosses $r(n)$ coins, makes $q(n)$ queries to (y, π), and outputs either 1 ("accept") or 0 ("reject").

Definition 3.14. *Let $s, \delta \in [0,1]$ and $r, q \colon \mathbb{N} \to \mathbb{N}$. A pair language L belongs to the class $\mathsf{PCP}_{s,\delta}[r(n), q(n)]$ if there exists a (r,q)-PCPP verifier V_L such that the following holds:*

1. **(Perfect) Completeness:** *If $(x, y) \in L$ then there exists π such that $\Pr_R[V_L^{(y,\pi)}(x;R) = 1] = 1$.*
2. **Soundness:** *If $\Delta(y, L_x) \geq \delta$ then for every π it holds that $\Pr_R[V_L^{(y,\pi)}(x;R) = 1] \leq 1 - s$.*

We call δ the **proximity parameter**.

We also consider a stronger notion of PCPPs, where the probability of rejection is proportional to the distance of y from L_x.

Definition 3.15. *Let $s \in (0,1] \times \mathbb{N} \to (0,1]$ and $r, q \colon \mathbb{N} \to \mathbb{N}$. A pair language L belongs to the class $\mathsf{Strong\text{-}PCP}_{s(\delta,n)}[r(n), q(n)]$ if there exists a (r,q)-PCPP verifier V_L such that the following holds:*

1. **(Perfect) Completeness:** *If $(x, y) \in L$ then there exists π such that $\Pr_R[V_L^{(y,\pi)}(x;R) = 1] = 1$.*
2. **Soundness:** *For every π it holds that $\Pr_R[V_L^{(y,\pi)}(x;R) = 1] \leq 1 - s(\Delta(y, L_x), n)$.*

Complexity parameters Throughout we will denote by $\mathsf{rand}(\cdot)$ the randomness complexity, by $\mathsf{query}(\cdot)$ the query complexity, and by $\mathsf{length}(\cdot)$ the proof length of the various PCPs and PCPPs that we will consider.[5] (The arguments to these functions will change from verifier to verifier, but will be given explicitly each time, and will be clear what they are.)

[5] More precisely, for PCPs of Proximity, the length will *not* account for the length of the object being tested.

4 Proof of Theorem 2

In this section we prove Theorem 2, which was discussed and formally stated in Section 2.3. The theorem states the existence of a PCPP system (cf. Definition 2.5) for the code ensemble $\mathcal{E}_{\mathrm{RS}}$ (cf. Definition 2.9) having a concrete-efficiency threshold $\hat{B}_{\mathrm{RS}} \leq 2^{35}$ (cf. Definition 2.7). Recall that our proof strategy is in two steps; we describe these two steps in Section 4.1 and Section 4.2 respectively. We then concisely summarize the proof of the theorem in Section 4.3. Throughout, it will be useful to keep in mind the definitions from Section 3.

4.1 Step 1: Improving the Universal Constant of Bivariate Testing

The first step in our proof of Theorem 2 is to improve the value of the *universal constant of bivariate testing*. Let us explain.

Testing proximity to bivariate low-degree polynomials via a "random row or column test" is at the basis of the soundness analysis of the PCPP verifier for Reed-Solomon codes of Ben-Sasson and Sudan [BSS08] as well as of its generalization constructed and analyzed in this paper. Specifically, the soundness analysis of [BSS08] relied on the following fact:

Lemma 4.1 ([BSS08, Lemma 6.12])**.** *Let $\mathbb{F}$ be a field. There exists a universal constant c_0 such that the following holds: for every two finite subsets A and B of $\mathbb{F}$, every two $d, e \geq 0$ with $d \leq |A|/4$ and $e \leq |B|/8$, and every function $g\colon A \times B \to \mathbb{F}$, it is the case that*

$$\delta^{(d,e)}_{A\times B}(g) \leq c_0 \cdot \left(\delta^{(d,*)}_{A\times B}(g) + \delta^{(*,e)}_{A\times B}(g)\right) \quad .$$

Recall that $\delta^{(d,*)}_{A\times B}(f)$, $\delta^{(*,e)}_{A\times B}(f)$, and $\delta^{(d,e)}_{A\times B}(f)$ respectively denote the distance of the function f to bivariate polynomial evaluations over $A \times B$ with degree in x at most d (and arbitrary degree in y), degree in y at most e (and arbitrary degree in x), and degree in x at most d and in y at most e. (See Definition 3.9.)

In other words, Lemma 4.1 says that testing proximity of a function $g\colon A \times B \to \mathbb{F}$ to $\mathrm{RM}\big(\mathbb{F}, A\times B, (d,e)\big)$, which is the Reed-Muller code over $A\times B$ of degree (d,e) (cf. Definition 3.6), can be done by examining the values of g over a random line in the first coordinate (i.e., a random "column") and over a random line in the second coordinate (i.e., a random "row"). The lemma is a corollary to the Bivariate Testing Theorem of Polishchuk and Spielman [PS94][Spi95, Theorem 4.2.19].

The crucial aspect of Lemma 4.1 that ultimately allows for the construction of quasilinear-size proximity proofs in [BSS08] (via a recursive composition) is the fact that the *fractional degree* (i.e., the ratio of the degree over the domain size) in both coordinates is constant, that is, the domain size need only be a constant larger than the degree — this is unlike the corresponding theorem in [AS98] where the size of the domain had to quadratically depend on the degree.

Another aspect of Lemma 4.1, which is very important from our perspective of concrete efficiency, is the value of the **universal constant of bivariate testing** c_0; specifically, the smaller a value one can show for c_0 the better soundness one can show for testing proximity to the Reed-Solomon code and, thus, ultimately construct PCPs with better soundness. (We shall discuss this in more detail shortly, in Section 4.2.) In [BSS08], it was shown that one can take $c_0 = 128$.

We show that one can take $c_0 = 10.24$. In fact, we state and prove the (improved) lemma of Ben-Sasson and Sudan in a more general form, letting c_0 depend on the fractional degree of each coordinate.

Theorem 4.2 (improved universal constant of bivariate testing)**.** *Let $\mathbb{F}$ be a field. Let $d, m, e, n \in \mathbb{N}$ be such that $1 > \frac{d}{m} + \frac{e}{n} + 2\delta$ for some positive δ. Define*

$$c_0 := \max\left\{3, \inf\left\{\delta^{-2} \,:\, \delta > 0 \text{ and } 1 > \tfrac{d}{m} + \tfrac{e}{n} + 2\delta\right\}\right\} .$$

Then, for every two finite subsets A and B of $\mathbb{F}$ of respective sizes m and n and every function $g\colon A \times B \to \mathbb{F}$, it is the case that

$$\delta^{(d,e)}_{A\times B}(g) \leq c_0 \cdot \left(\delta^{(d,*)}_{A\times B}(g) + \delta^{(*,e)}_{A\times B}(g)\right) .$$

By setting $\frac{d}{m} := \frac{1}{4}$ and $\frac{e}{n} := \frac{1}{8}$ as in Lemma 4.1, we recover the (improved) constant $c_0 = 10.24$, as opposed to the previous value $c_0 = 128$.

The proof of the theorem is given in Section 5; it follows via improvements to the proof of [Spi95, Theorem 4.2.19] and its corollary Lemma 4.1.

The above lemma can now also be invoked with different settings of $\frac{d}{m}$ and $\frac{e}{n}$, other than the $\frac{d}{m} = \frac{1}{4}$ and $\frac{e}{n} = \frac{1}{8}$ originally used in Lemma 4.1. Different settings of $\frac{d}{m}$ and $\frac{e}{n}$, eventually correspond (as can be seen by inspecting how Theorem 4.2 is eventually invoked in the soundness analysis in Section 6) to changing the "rate" of the proof of proximity; for example, with $\frac{d}{m} = \frac{1}{8}$ and $\frac{e}{n} = \frac{1}{8}$, we obtain the smaller constant $c_0 = 64/9 \approx 7.1$ at the cost of decreasing the rate. More generally, it is possible to decrease c_0 further if one is willing to decrease either the ratio $\frac{d}{m}$ or $\frac{e}{n}$ (or both); doing so is possible, but will increase the length of the proof of proximity (i.e., decreasing the rate). The point is that the lemma stated in this way allows us to easily deduce what constant c_0 to use for different choices of rate; it will also enable us to find optimal choices of parameters for a given concrete problem size. Explicitly having these degrees of freedom will ultimately be crucial to enable us to find find a construction with a good concrete-efficiency threshold.

We believe that Theorem 4.2 can be further improved, and we believe it is an important open problem to do so: even small improvements (e.g., relaxing the requirement in the theorem from $1 > \frac{d}{m} + \frac{e}{n} + 2\delta$ to $2 > \frac{d}{m} + \frac{e}{n} + 2\delta$) eventually translate into tremendous improvements to the concrete-efficiency thresholds of PCPP systems for Reed-Solomon codes.

4.2 Step 2: Improving the Soundness Analysis of the Recursive Construction

In general, the verifier of a PCPP system consists of the sequential repetition of a simpler "basic" verifier, and the number of repetitions is inversely proportional to the *soundness* of this basic verifier; the soundness may be a function of various parameters. The second step in our proof of Theorem 2 is to provide a class of constructions that generalize the "basic" verifiers of [BSS08] and deduce for this class a much tighter and explicit soundness analysis. Obtaining as good a soundness as possible is critical for improving concrete efficiency, because worse (i.e., lower) soundness directly translates into a greater number of repetitions needed to bring the soundness up to, say, $1/2$; each repetition costs more queries, random coins, and, most importantly, running time.

Concretely, the second step of our theorem consists of much tighter soundness analyses for generalized variants of the (strong) PCPP verifiers (cf. Definition 3.15) testing proximity to RS and VRS (cf. Definition 3.3 and Definition 3.4) constructed by Ben-Sasson and Sudan [BSS08] based on Lemma 4.1; we respectively denote these two PCPP verifiers by V_{RS} and V_{VRS}. While of course in our case we start with a "stronger and more general foundation" (namely, our Theorem 4.2 that strengthens and generalizes Lemma 4.1, as discussed in Section 4.1), we still seek a much more careful (and generalized) analysis of the soundness of the recursive algorithms underlying V_{RS} and V_{VRS}. We indeed do so and our end result is formally given in Theorem 6.1 and proved in Section 6; we now discuss at high level what the proof of this theorem involves.

The focus of our effort is improving the soundness of $V_{\mathrm{RS},=}$, which is the PCPP verifier "at the bottom of the stack" responsible for testing proximity to $\mathrm{RS}(\mathbb{F}, L, |L|/8)$, for a subspace L of $\mathbb{F}$, and is where a good part of the technical work of [BSS08] is carried out. Indeed, V_{VRS} can be constructed based on V_{RS}; V_{RS} invokes one of three PCPP verifiers, $V_{\mathrm{RS},>}$, $V_{\mathrm{RS},<}$, or $V_{\mathrm{RS},=}$, depending on whether the fractional degree to be tested is respectively greater than, less that, or equal to $1/8$; both of the PCPP verifiers $V_{\mathrm{RS},>}$ and $V_{\mathrm{RS},<}$ consist of few invocations of $V_{\mathrm{RS},=}$. So, again, the technical heart of the construction of V_{RS} and V_{VRS} is $V_{\mathrm{RS},=}$ and, while we also take the care to tighten the soundness analysis of $V_{\mathrm{RS},<}$, $V_{\mathrm{RS},>}$, V_{RS}, V_{VRS}, improving the soundness analysis of $V_{\mathrm{RS},=}$ is where most of our effort goes.

The relevant lemma for $V_{\mathrm{RS},<}$ from [BSS08] is:

Lemma 4.3 ([BSS08, Lemma 6.10])**.** *There exists a constant $c \geq 1$ such that for every positive integer κ and every positive ε the following holds: if for a function $p\colon S \to \mathbb{F}_{2^\ell}$, a string π, an integer ℓ, and a κ-dimensional linear subspace $L \subseteq \mathbb{F}_{2^\ell}$ it holds that*

$$\Pr\left[V_{\mathrm{RS},=}^{(p,\pi)}\big(\mathbb{F}_{2^\ell}, L, |L|/8 - 1\big) = 1\right] > 1 - \varepsilon \ ,$$

then

$$\Delta_L\Big(p, \mathrm{RS}\big(\mathbb{F}_{2^\ell}, L, |L|/8 - 1\big)\Big) \leq c^{\log \kappa} \cdot \varepsilon \ .$$

Recall that Δ_L denotes the fractional Hamming distance for strings/functions defined over the set (in this case, subspace) L. (See Definition 3.7.) Also recall that a PCPP verifier is given as input an *explicit input* and is granted oracle access to an *implicit input* and a *proximity proof*; in the case of $V_{\mathrm{RS},=}$, the explicit input specifies the desired Reed-Solomon code and the implicit input is the function p that needs to be tested, with the aid of the proximity proof π.

The above lemma says that the (strong) PCPP verifier $V_{\mathrm{RS},=}$ has a soundness function $s_{\mathrm{RS},=}$ (cf. Definition 3.15) that can be lower bounded as follows:

$$s_{\mathrm{RS},=}(\delta, n) \geq \frac{\delta}{\kappa^{\log c}} \ .$$

We note that one cannot hope in a soundness analysis showing that $s_{\mathrm{RS},=}(\delta, n) \geq \kappa^{-o(1)}$, because $V_{\mathrm{RS},=}$ consists of $O(\log \log n) = O(\log \kappa)$ recursions of a constant-soundness low-degree test (namely, the "random row or column test"), so that we indeed expect $s_{\mathrm{RS},=}(\delta, n) \leq \kappa^{-O(1)}$. Therefore, the parameter of interest here is the constant c, and thus our goal of obtaining a better soundness analysis for $V_{\mathrm{RS},=}$ can now be technically restated as:

find the **smallest** *possible c such that Lemma 4.3 holds.*

Ben-Sasson and Sudan proved Lemma 4.3 for $c = \big(64(c_0 + 2/3)\big)^{\frac{1}{\log(7/6)}}$ and $c_0 = 128$ (and note that, as already mentioned, the universal constant of bivariate testing c_0 appears as part of the soundness analysis of $V_{\mathrm{RS},=}$); this yields $\log c \approx 58.5$, which is clearly an impractical value. Indeed, with such an analysis, one can only show that $\hat{B}_{\mathrm{RS}} \leq 2^{572}$. (Since values of c are quite large, we shall always give its *binary logarithm* when computing bounds for it.)

Without modifying the original construction, we are able to improve the soundness analysis to show a constant c with $\log c \approx 23.2$ as follows:

1. By using the improved universal constant of bivariate testing $c_0 = 10.24$ (by invoking our Theorem 4.2) in the old soundness analysis of $V_{\mathrm{RS},=}$ showing $c = \big(64(c_0 + 2/3)\big)^{\frac{1}{\log(7/6)}}$ we get $\log c \approx 42.5$.

2. We perform the soundness analysis of $V_{\mathrm{RS},=}$ much more carefully, yielding the improved expression for $c = \big(\frac{16}{3}(1 + c_0) - 2\sqrt{16 + \frac{64}{3}(1 + c_0)} + 8\big)^{\frac{1}{\log(7/6)}}$ so that, using $c_0 = 10.24$, we get $\log c \approx 23.2$.

Unfortunately, $\log c \approx 23.2$ is still far from practicality. We attempt to go further by *fine tuning* details of the construction of $V_{\mathrm{RS},=}$, and we are able to obtain $\log c \approx 10.6$ with a slightly different construction (but the same complexity parameters — in time, proof length, and number of queries — as the original).

Even with $\log c \approx 10.6$, however, we are still far from practicality, and need to work more. And this is also where our technical approach to studying concrete efficiency of PCPP verifiers departs from "standard" approaches used to study their asymptotic efficiency. We proceed in two steps:

- We develop an analysis that lets us efficiently explore a class of $V_{\mathrm{RS},=}$ constructions related to the original one of Ben-Sasson and Sudan. To that end, we explicitly parametrize several quantities in the construction of $V_{\mathrm{RS},=}$ (namely, the fractional degree to be tested, the query complexity, and how the recursion is "balanced") and, with the ultimate goal of deriving a precise understanding of how these parameters trade off against each other, *we are able to deduce an explicit expression for the constant c in terms of these parameters*. We are then in a position to identify better tradeoffs among these parameters for any given problem size.

  Concretely, see Theorem 6.3 (and Corollary 6.4) in Section 6.1 for the formal statement giving the expression for c in terms of four parameters $(\eta, \kappa_0, \gamma, \mu)$, where $2^{-\eta}$ denotes the fractional degree, 2^{κ_0} the number of queries, γ how the subspace L is split for the recursion and μ what is the "overlap" between the two subspaces of L in the recursion. In the original construction, $\eta = 3$, $\kappa_0 = 6$, $\gamma = 0$, $\mu = 2$.

  (Our improved analysis also enables us to choose the larger fractional degree $2^{-\eta}$ with $\eta = 2$, whereas previously the analysis was not strong enough to show any soundness for this setting, which yields an even shorter proximity proof.)

  As an example, in Table 1, we give values for $\log c$ as we change η and κ_0 (for optimal choices of the other parameters in each case).

- We leverage our generalized soundness analysis to derive a recursive function that lets us *numerically compute* the "effective soundness" for any given problem size. We discover that effective soundness is a function that approaches very slowly our theoretical analysis (which

$\log c$	$\kappa_0 = 6$	$\kappa_0 = 7$	$\kappa_0 = 8$	$\kappa_0 = 9$	$\kappa_0 = 10$
$\eta = 2$	18.2	13.0	15.1	12.0	13.6
$\eta = 3$	**10.6**	12.5	8.8	10.0	7.9
$\eta = 4$	16.3	18.8	10.0	11.3	7.9
$\eta = 5$	11.4	13.2	15.0	16.7	8.8

Table 1: Trade-offs in the choice of parameters for $V_{\mathrm{RS},=}$: increasing η and κ_0 respectively increases the PCP oracle length and the query complexity; the value $\log c \approx 10.6$ for the setting $\eta = 3$ and $\kappa_0 = 6$ (in bold) corresponds to what we obtain for (a fine tuning of) the original construction, improving on the original $\log c \approx 58.5$.

holds for all problem sizes), and thus we are able to *derive a much better "concrete" lower bound*, where "concrete" means for any problem size up to 2^{100}. See Section 6.1.3 (specifically, Equation 26 and Equation 27) for our result. As an example, in Table 2, we give values for the "effective $\log c$" for the same choices of parameters that we used in Table 1; we see great improvements throughout the table.

$\log c$	$\kappa_0 = 6$	$\kappa_0 = 7$	$\kappa_0 = 8$	$\kappa_0 = 9$	$\kappa_0 = 10$
$\eta = 2$	8.1	6.8	7.0	6.0	6.1
$\eta = 3$	**5.3**	4.8	4.6	4.4	4.0
$\eta = 4$	5.7	4.9	4.3	4.1	3.9
$\eta = 5$	5.3	4.4	5.2	4.5	4.0

Table 2: Values of the "effective" $\log c$ for practical problem sizes, for different choices of parameters for $V_{\mathrm{RS},=}$. For the setting $\eta = 3$ and $\kappa_0 = 6$ (in bold), $\log c$ improved from the theoretical value 10.6 to the effective value 5.3.

4.3 Step 3: Putting Things Together

We now explain how the two steps described in the previous two subsections come together to give a proof of Theorem 2:

Proof of Theorem 2. The PCPP verifier $V_{\mathrm{RS},=}$ with a choice of parameters a choice of parameters $(\eta, \kappa_0, \gamma, \mu)$ tests proximity to the ensemble

$$\mathcal{E}^{(\eta)} \stackrel{\text{def}}{=} \left\{ \mathrm{RS}(\mathbb{F}_{2^\ell}, L, d) \;\middle|\; L \text{ is a subspace of } \mathbb{F}_{2^\ell}\,,\; d/|L| = 2^{-\eta} \right\}\,. \tag{4}$$

The code ensemble considered in Theorem 2 is $\mathcal{E}_{\mathrm{RS}} = \mathcal{E}^{(3)}$; any code $C \in \mathcal{E}_{\mathrm{RS}}$ is thus fully specified by ℓ and L. Recall that

$$\hat{B}_{\mathrm{RS}} \stackrel{\text{def}}{=} \arg\min_{\hat{B}' \in \mathbb{N}} \left\{ \hat{B}' \;\middle|\; \forall\ [n,k,d]_{2^\ell}\text{-code } C \in \mathcal{E}_{\mathrm{RS}} \text{ with } k \geq \hat{B}'\,,\ \left(\frac{n + L(C)}{k}\right) \cdot Q(C) < \frac{1}{2} \cdot k \right\}\,,$$

where $L(C)$ is the length of the PCPP proof for codewords in C and $Q(C)$ is the query complexity for testing proximity (with soundness $1/2$) of non-codewords that are sufficiently far from

C. By examining $V_{\mathrm{RS},=}$ (see Lemma B.1 for details), we deduce that for every $C \in \mathcal{E}_{\mathrm{RS}}$, letting $\kappa := \dim(L)$,

- $L(C) \leq 2^{\kappa} \cdot \kappa^{1+\max\{-\gamma+\mu+1,\gamma\}}$,
- $Q(C) \leq 2^{\kappa_0} \cdot \kappa^{\log c(\eta,\kappa_0,\gamma,\mu)}$.[6]

Thus, recalling that $n = |L| = 2^{\kappa}$, if

$$(1 + \kappa^{1+\max\{-\gamma+\mu+1,\gamma\}}) \cdot 2^{\kappa_0} \cdot \kappa^{\log c(\eta,\kappa_0,\gamma,\mu)} < \frac{1}{2} \cdot 2^{\kappa} \tag{5}$$

then

$$\left(\frac{n + L(C)}{k}\right) \cdot Q(C) < \frac{1}{2} \cdot k \ .$$

Then, Theorem 6.3 tells us a bound on $\log c(\eta, \kappa_0, \gamma, \mu)$, so that we can now compute a bound on $\hat{B}_{\mathrm{RS}}$ by using Equation 5. In fact, we can numerically compute $L(C)$ exactly and, moreover, the proof of Theorem 6.3 enables us to numerically evaluate a bound on $\log c(\eta, \kappa_0, \gamma, \mu)$ (as discussed in Section 6.1.3); we do so in our computations to obtain an even better bound on $\hat{B}_{\mathrm{RS}}$.

For example, using the parameters $(\eta, \kappa_0, \gamma, \mu) = (3, 14, 1, 2)$, we get $\hat{B}_{\mathrm{RS}} \leq 2^{35}$, as claimed. (In contrast, using the original parameters $(\eta, \kappa_0, \gamma, \mu) = (3, 6, 0, 2)$ and the bound $\log c \approx 58.5$ obtained in [BSS08], we only get $\hat{B}_{\mathrm{RS}} \leq 2^{572}$.) □

[6] Note that the number of queries 2^{κ_0} has been multiplied by $\kappa^{\log c(\eta,\kappa_0,\gamma,\mu)}$ to take into account the fact that the verifier that has been amplified to achieve soundness $1/2$; see Remark 6.2.

5 Improved Universal Constant of Bivariate Testing

After recalling some basic lemmas about polynomials from [Spi95] (Section 5.1), we prove Theorem 4.2 in two steps (respectively in Section 5.2 and Section 5.3); as an example, we then show how to obtain the improved universal constant of bivariate testing $c_0 = 10.24$ (Section 5.4).

5.1 Some Basic Lemmas

Let us recall some basic lemmas needed for the proof of the theorem.

Lemma 5.1 ([Spi95, Proposition 4.2.9]). *Let $A = \{\alpha_1, \ldots, \alpha_m\} \subseteq \mathbb{F}$, $B = \{\beta_1, \ldots, \beta_n\} \subseteq \mathbb{F}$, $d < m$, and $e < n$. Let $f\colon A \times B \to \mathbb{F}$ be a function such that for every $\alpha \in A$ it holds that $f(\alpha, y)$ agrees with some polynomial over $\mathbb{F}$ of degree d and for every $\beta \in B$ it holds that $f(x, \beta)$ agrees with some polynomial over $\mathbb{F}$ of degree e. Then there exists a polynomial $P(x, y)$ over $\mathbb{F}$ of degree (d, e) such that f agrees with P everywhere on $A \times B$.*

Proof sketch. By "stringing together" $e + 1$ low-degree row polynomials via Lagrange interpolation in one variable. □

Lemma 5.2 ([Spi95, Lemma 4.2.13]). *Let $\mathbb{F}$ be a field, $A = \{\alpha_1, \ldots, \alpha_m\} \subseteq \mathbb{F}$, and $B = \{\beta_1, \ldots, \beta_n\} \subseteq \mathbb{F}$. Let $S \subseteq A \times B$ be a set of size at most $\delta^2 mn$. Then there exists a non-zero polynomial $E(x, y)$ over $\mathbb{F}$ of degree $(\delta m, \delta n)$ such that $E(\alpha, \beta) = 0$ for all $(\alpha, \beta) \in S$.*

Proof sketch. The "evaluation map" is a linear operator; in this case, the domain is a vector space of dimension $(\lfloor \delta m \rfloor + 1)(\lfloor \delta n \rfloor + 1)$ and the range is a vector space of dimension $\delta^2 mn$; hence the kernel contains more than one element and, in particular, a non-zero solution. □

Lemma 5.3 ([Spi95, Lemma 4.2.14]). *Let $\mathbb{F}$ be a field, $A = \{\alpha_1, \ldots, \alpha_m\} \subseteq \mathbb{F}$, and $B = \{\beta_1, \ldots, \beta_n\} \subseteq \mathbb{F}$. Let $E(x, y), P(x, y), R(x, y), C(x, y)$ be polynomials over $\mathbb{F}$ of respective degrees $(\delta m, \delta n), (d + \delta m, e + \delta n), (d, n), (m, e)$ such that for every $(\alpha, \beta) \in A \times B$ it holds that $P(\alpha, \beta) = R(\alpha, \beta)E(\alpha, \beta) = C(\alpha, \beta)E(\alpha, \beta)$. If $|A| > d + \delta m$ then for all $\alpha \in A$ it holds that $E(\alpha, y)$ divides $P(\alpha, y)$. If $|B| > e + \delta n$ then for all $\beta \in B$ it holds that $E(x, \beta)$ divides $P(x, \beta)$.*

Proof sketch. For any $\alpha \in A$, $P(\alpha, y)$ and $R(\alpha, y)E(\alpha, y)$ agree on $|B| > e + \delta n$ points, and thus they are equal as formal polynomaials. Similarly for the other coordinate. □

Lemma 5.4 ([Spi95, Lemma 4.2.18]). *Let $\mathbb{F}$ be a field, $A = \{\alpha_1, \ldots, \alpha_m\} \subseteq \mathbb{F}$, and $B = \{\beta_1, \ldots, \beta_n\} \subseteq \mathbb{F}$. Let $E(x, y)$ and $P(x, y)$ be polynomials over $\mathbb{F}$ of respective degrees $(\delta m, \varepsilon n)$ and $(\delta' m + \delta m, \varepsilon' n + \varepsilon n)$. Suppose that for every $\alpha \in A$ and $\beta \in B$ it holds that $E(\alpha, y)$ divides $P(\alpha, y)$ and $E(x, \beta)$ divides $P(x, \beta)$. If $1 > \delta' + \varepsilon' + \delta + \varepsilon$ then $E(x, y)$ divides $P(x, y)$.*

Proof sketch. The proof is non-trivial; it uses resultants and their derivatives. (An alternate proof using dimensions of vector spaces can be found in [Sze05].) □

5.2 The PS Bivariate Testing Theorem

We begin by stating and proving the Polishchuk Spielman Bivariate Testing Theorem with our slightly improved parameters:

Theorem 5.5. *Let $\mathbb{F}$ be a field, $A = \{\alpha_1, \ldots, \alpha_m\} \subseteq \mathbb{F}$, and $B = \{\beta_1, \ldots, \beta_n\} \subseteq \mathbb{F}$. Let $R(x,y)$ be a polynomial over $\mathbb{F}$ of degree (d,n) and let $C(x,y)$ be a polynomial over $\mathbb{F}$ of degree (m,e). Suppose that*

$$\Pr_{(\alpha,\beta)\in A\times B}[R(\alpha,\beta) \neq C(\alpha,\beta)] < \delta^2 \text{ and } 1 > \frac{d}{m} + \frac{e}{n} + 2\delta \ .$$

Then there exists a polynomial Q over $\mathbb{F}$ of degree (d,e) such that

$$\Pr_{(\alpha,\beta)\in A\times B}[R(\alpha,\beta) \neq Q(\alpha,\beta) \text{ or } C(\alpha,\beta) \neq Q(\alpha,\beta)] < 2\delta^2 \ .$$

The improvement is in that we relax the original requirement $1 > 2\left(\frac{d}{m} + \frac{e}{n} + \delta\right)$ to the new requirement $1 > \frac{d}{m} + \frac{e}{n} + 2\delta$, by simply being more careful in the original proof and combining it with another proof of the theorem appearing in [Sze05, Theorem 5.30].

Proof of Theorem 5.5. Let $S = \{(\alpha,\beta) \in A \times B \text{ s.t. } R(\alpha,\beta) \neq C(\alpha,\beta)\}$; by assumption, $|S| < \delta^2 mn$. By Lemma 5.2, there exists a (non-zero) polynomial $E(x,y)$ over $\mathbb{F}$ of degree $(\delta m, \delta n)$ such that $E(S) = \{0\}$. Note that $R(x,y)E(x,y)$ is a polynomial of degree $(d+\delta m, n+\delta n)$ and $C(x,y)E(x,y)$ is a polynomial of degree $(m+\delta m, e+\delta n)$.

By Lemma 5.1, there exists a polynomial $P(x,y)$ over $\mathbb{F}$ of degree $(d+\delta m, e+\delta n)$ such that for every $(\alpha,\beta) \in A \times B$ it holds that $P(\alpha,\beta) = R(\alpha,\beta)E(\alpha,\beta) = C(\alpha,\beta)E(\alpha,\beta)$.

By hypothesis we know that $m > d+\delta m$ and $n > e+\delta n$; therefore by Lemma 5.3 we deduce that for all $\alpha \in A$ it holds that $E(\alpha,y)$ divides $P(\alpha,y)$ and for all $\beta \in B$ it holds that $E(x,\beta)$ divides $P(x,\beta)$.

By hypothesis we know that $1 > \delta' + \varepsilon' + \delta + \varepsilon$, where we define $\delta' := \frac{d}{m}$, $\varepsilon' := \frac{e}{n}$, $\varepsilon := \delta$; therefore by Lemma 5.4 we deduce that $E(x,y)$ divides $P(x,y)$.

So define $Q(x,y) := P(x,y)/E(x,y)$, and note that $Q(x,y)$ is of degree (d,e) and for every $(\alpha,\beta) \in A \times B$ it holds that $Q(\alpha,\beta)E(\alpha,\beta) = R(\alpha,\beta)E(\alpha,\beta) = C(\alpha,\beta)E(\alpha,\beta)$.

Let $T = \{(\alpha,\beta) \in A \times B \text{ s.t. } R(\alpha,\beta) = C(\alpha,\beta) \neq Q(\alpha,\beta)\}$. Note that

$$\left\{(\alpha,\beta) \in A \times B \text{ s.t. } R(\alpha,\beta) \neq Q(\alpha,\beta) \text{ or } C(\alpha,\beta) \neq Q(\alpha,\beta)\right\} \subseteq T \cup S \ .$$

We now argue that $|T| \leq \delta^2 mn$. (And doing so will complete the proof, since $|S| < \delta^2 mn$ by assumption.)

Indeed, assume by way of contradiction that $|T| > \delta^2 mn$. Then either $A_1 := \{\alpha \in A \text{ s.t. } \exists\, \beta \in B \text{ with } (\alpha,\beta) \in T\}$ has size greater than δm or $B_1 := \{\beta \in B \text{ s.t. } \exists\, \alpha \in A \text{ with } (\alpha,\beta) \in T\}$ has size greater than δn.

Suppose that $|A_1| > \delta m$. Fix any $\alpha \in A_1$. Note that $R(\alpha,y)$ and $Q(\alpha,y)$ are distinct polynomials. However, we know that $Q(\alpha,y)E(\alpha,y)$ and $R(\alpha,y)E(\alpha,y)$ agree everywhere on B. Since $|B| = n > e + \delta n \geq \deg_y Q + \deg_y E$, we deduce that $E(\alpha,y)$ is identically zero, and thus $(x-\alpha)$ is a factor of E. We conclude that $\prod_{\alpha\in A_1}(x-\alpha)$ divides E, but this is a contradiction because $|A_1| > \delta m$ and $\deg_x E \leq \delta m$.

Suppose that $|B_1| > \delta n$ instead. We can then argue in the same way. Namely, fix any $\beta \in B_1$. Note that $R(x,\beta)$ and $Q(x,\beta)$ are distinct polynomials. However, we know that $Q(x,\beta)E(x,\beta)$ and $R(x,\beta)E(x,\beta)$ agree everywhere on A. Since $|A| = m > d + \delta m \geq \deg_x Q + \deg_x E$, we deduce that $E(x,\beta)$ is identically zero, and thus $(y-\beta)$ is a factor of E. We conclude that $\prod_{\beta\in B_1}(y-\beta)$ divides E, but this is a contradiction because $|B_1| > \delta n$ and $\deg_y E \leq \delta n$. □

5.2.1 Barriers to further improvements

It is not clear whether one can relax further the requirement that $1 > \frac{d}{m} + \frac{e}{n} + 2\delta$ in Theorem 5.5. So let us briefly discuss how this requirement arises in the proof of Theorem 5.5. There are three main places:

(i) Having found E such that $P(\alpha, \beta) = R(\alpha, \beta)E(\alpha, \beta) = C(\alpha, \beta)E(\alpha, \beta)$ for every $(\alpha, \beta) \in A \times B$, we use the fact that $1 > \frac{d}{m} + \delta$ and $1 > \frac{e}{n} + \delta$ to deduce (via Lemma 5.3) that for every $\alpha \in A$ and $\beta \in B$ it holds that $E(\alpha, y)$ divides $P(\alpha, y)$ and $E(x, \beta)$ divides $P(x, \beta)$.

(ii) Then, we use the fact that $1 > \frac{d}{m} + \frac{e}{n} + 2\delta$ to deduce (via Lemma 5.4) that $E(x, y)$ divides $P(x, y)$.

(iii) Later in the proof, we use (again) the fact the $1 > \frac{d}{m} + \delta$ and $1 > \frac{e}{n} + \delta$, to deduce that $|T| \leq \delta^2 mn$.

The requirement in (ii) is the more stringent one and, in our view, is the one that one should be able to relax further. We now briefly describe prospects of improving it.

A simple limit to Lemma 5.4. In the proof of Theorem 5.5 we are given the set of "errors" S (i.e., those places where the row polynomial R differs from the column polynomial C), and need to find an error-locator polynomial E vanishing on S for which we can show that E divides the induced P. Roughly, the way this is currently done is to ensure that E has low-degree in both variables (namely, δm and δn respectively, enough to ensure that one always exists) and then invoking the "Bézout-type" argument of Lemma 5.4.

It is easy to note, however, that the requirement $1 > \delta' + \varepsilon' + \delta + \varepsilon$ in Lemma 5.4 (which translates into the requirement $1 > \frac{d}{m} + \frac{e}{n} + 2\delta$ of (ii)) cannot be improved beyond $2 > \delta' + \varepsilon' + \delta + \varepsilon$ (which would translate into the relaxed requirement $2 > \frac{d}{m} + \frac{e}{n} + 2\delta$). Indeed, consider the choice of polynomials

$$P(x, y) = \left(\prod_{i=1}^{m}(x - \alpha_i)\right)\left(\prod_{i=1}^{n}(y - \beta_i)\right) \quad \text{and} \quad E(x, y) = x - y \ ;$$

even though E does divide P in every row and column (as P is zero), it is *not* the case that E divides P. (In this example, we would have $\delta = \frac{1}{m}$, $\varepsilon = \frac{1}{n}$, $\delta' = \frac{m-1}{m}$, and $\varepsilon' = \frac{n-1}{n}$, in which case $\delta' + \varepsilon' + \delta + \varepsilon = 2$.)

Nonetheless, this observation is not discouraging at all, as improving Theorem 5.5 to a requirement such as $2 > \frac{d}{m} + \frac{e}{n} + 2\delta$ would be great, and in fact seems a reasonable goal.

5.3 The Universal Constant for Bivariate Testing

We re-prove Lemma 4.1 via a simple case analysis that is a slight refinement of the case analysis originally used in [BSS08, proof of Lemma 6.12]. Specifically, the two analyses differ in that we split the two cases according to a bound on the *sum* of $\delta^{(d,*)}_{A\times B}(g)$ and $\delta^{(*,e)}_{A\times B}(g)$, while [BSS08] split according to a bound on *each of the two* quantities; this improves the constant by a factor of two.

In fact, we state and prove the (improved) lemma of Ben-Sasson and Sudan in a more general form, by using the generic fractional degrees $\frac{d}{m}$ and $\frac{e}{n}$ (as opposed to $\frac{1}{4}$ and $\frac{1}{8}$ as in the original lemma).

Define $\Phi(\delta, d, m, e, n)$ to be a "δ-monotone" predicate denoting whether Theorem 5.5 holds for the proximity parameter $\delta > 0$ and fractional degrees $\frac{d}{m}$ and $\frac{e}{n}$.[7] (For example, we can take $1 \overset{?}{>} \frac{d}{m} + \frac{e}{n} + 2\delta$ as the predicate $\Phi(\delta, d, m, e, n)$.)

Lemma 5.6. *Let $\mathbb{F}$ be a field. Let $d, m, e, n \in \mathbb{N}$ be such that $\Phi(\delta, d, m, e, n) = 1$ for some positive δ. Then we can take*

$$c_0 := \max\left\{3, \inf\left\{\delta^{-2} \,:\, \delta > 0 \text{ and } \Phi(\delta, d, m, e, n) = 1\right\}\right\}$$

in Lemma 4.1. That is, for every two finite subsets A and B of $\mathbb{F}$ of respective sizes m and n and every function $g\colon A \times B \to \mathbb{F}$, it is the case that

$$\delta^{(d,e)}_{A\times B}(g) \le c_0 \cdot \left(\delta^{(d,*)}_{A\times B}(g) + \delta^{(*,e)}_{A\times B}(g)\right) \ .$$

Proof of Lemma 5.6. We distinguish between two cases:

Case 1. Suppose that $\delta^{(d,*)}_{A\times B}(g) + \delta^{(*,e)}_{A\times B}(g) < 1/c_0$. Fix any ε with $0 < \varepsilon < 1/c_0 - \delta^{(d,*)}_{A\times B}(g) - \delta^{(*,e)}_{A\times B}(g)$. Let $R(x, y)$ be the polynomial corresponding to a codeword in $\mathrm{RM}(\mathbb{F}, A \times B, (d, |B| - 1))$ closest to g; note that $\Delta_{A\times B}(g, R) = \delta^{(d,*)}_{A\times B}(g)$. By the triangle inequality,

$$\begin{aligned}
\delta^{(d,e)}_{A\times B}(g) &= \Delta_{A\times B}\big(g, \mathrm{RM}(\mathbb{F}, A \times B, (d, e))\big) \\
&\le \Delta_{A\times B}(g, R) + \Delta_{A\times B}\big(R, \mathrm{RM}(\mathbb{F}, A \times B, (d, e))\big) \\
&= \delta^{(d,*)}_{A\times B}(g) + \Delta_{A\times B}\big(R, \mathrm{RM}(\mathbb{F}, A \times B, (d, e))\big) \ .
\end{aligned}$$

Let $C(x, y)$ be the polynomial corresponding to a codeword in $\mathrm{RM}(\mathbb{F}, A \times B, (|A| - 1, e))$ closest to g; note that $\Delta_{A\times B}(g, C) = \delta^{(*,e)}_{A\times B}(g)$. By the triangle inequality,

$$\begin{aligned}
\delta^{(d,e)}_{A\times B}(g) &= \Delta_{A\times B}\big(g, \mathrm{RM}(\mathbb{F}, A \times B, (d, e))\big) \\
&\le \Delta_{A\times B}(g, C) + \Delta_{A\times B}\big(C, \mathrm{RM}(\mathbb{F}, A \times B, (d, e))\big) \\
&= \delta^{(*,e)}_{A\times B}(g) + \Delta_{A\times B}\big(C, \mathrm{RM}(\mathbb{F}, A \times B, (d, e))\big) \ .
\end{aligned}$$

For the choice $\delta_\varepsilon = \sqrt{\delta^{(d,*)}_{A\times B}(g) + \delta^{(*,e)}_{A\times B}(g) + \varepsilon} < \sqrt{1/c_0}$, note that the following two conditions hold:

$$\begin{aligned}
\Pr_{(\alpha,\beta)\in A\times B}\big[R(\alpha, \beta) \neq C(\alpha, \beta)\big] &= \Delta_{A\times B}(R, C) \\
&\le \Delta_{A\times B}(R, g) + \Delta_{A\times B}(g, C) \\
&= \delta^{(d,*)}_{A\times B}(g) + \delta^{(*,e)}_{A\times B}(g) \\
&< \delta^2_\varepsilon
\end{aligned}$$

and

$$\Phi(\delta_\varepsilon, d, m, n, e) = 1 \ .$$

[7] By "δ-monotone" we mean that if $\Phi(\delta, d, m, e, n) = 1$ for some δ then for any $\delta' \in (0, \delta)$ we also have $\Phi(\delta', d, m, e, n) = 1$.

(Indeed, letting $\bar{\delta} := \sup\{\delta : \delta > 0 \text{ and } \Phi(\delta, d, m, e, n) = 1\}$, we see that $c_0 = \max\{3, 1/\bar{\delta}^2\}$. Since $\delta_\varepsilon < \frac{1}{\sqrt{c_0}}$, we deduce that $\delta_\varepsilon < \frac{1}{\sqrt{\max\{3,1/\bar{\delta}^2\}}} \leq \frac{1}{\sqrt{1/\bar{\delta}^2}} = \bar{\delta}$, so that $\Phi(\delta_\varepsilon, d, m, n, e) = 1$ as we just claimed.)

Hence, invoking Theorem 5.5 with the $\mathbb{F}$, A, B, d, m, e, n, R, and C of this proof, and using $\delta = \delta_\varepsilon$, we deduce that $\Delta_{A\times B}\big(R, \mathrm{RM}(\mathbb{F}, A \times B, (d, e))\big) < 2\delta_\varepsilon^2$ and $\Delta_{A\times B}\big(C, \mathrm{RM}(\mathbb{F}, A \times B, (d, e))\big) < 2\delta_\varepsilon^2$; combining with the two upperbounds on $\delta^{(d,e)}_{A\times B}(g)$ derived earlier, we obtain

$$\delta^{(d,e)}_{A\times B}(g) = \frac{\delta^{(d,e)}_{A\times B}(g)}{2} + \frac{\delta^{(d,e)}_{A\times B}(g)}{2} \leq \frac{\delta^{(d,*)}_{A\times B}(g)}{2} + \frac{\delta^{(*,e)}_{A\times B}(g)}{2} + 2\delta_\varepsilon^2 = \frac{5\delta^{(d,*)}_{A\times B}(g)}{2} + \frac{5\delta^{(*,e)}_{A\times B}(g)}{2} + \varepsilon \ .$$

Choosing ε small enough, we get $\delta^{(d,e)}_{A\times B}(g) \leq 3 \cdot \Big(\delta^{(d,*)}_{A\times B}(g) + \delta^{(*,e)}_{A\times B}(g)\Big)$.

Case 2. Suppose that $\delta^{(d,*)}_{A\times B}(g) + \delta^{(*,e)}_{A\times B}(g) \geq 1/c_0$. Then the upperbound on $\delta^{(d,e)}_{A\times B}(g)$ holds trivially:

$$\delta^{(d,e)}_{A\times B}(g) \leq 1 = c_0 \cdot \frac{1}{c_0} \leq c_0 \cdot \Big(\delta^{(d,*)}_{A\times B}(g) + \delta^{(*,e)}_{A\times B}(g)\Big) \ .$$

In either case, we have shown the desired upper bound on $\delta^{(d,e)}_{A\times B}(g)$. □

5.4 Putting Things Together

Combining Theorem 5.5 and Lemma 5.6, we obtain Theorem 4.2.

For example, by using the best predicate $\Phi(\delta, d, m, e, n)$ we could prove (namely, $1 \stackrel{?}{>} \frac{d}{m} + \frac{e}{n} + 2\delta$ as the predicate $\Phi(\delta, d, m, e, n)$ from Theorem 5.5) and by setting $\frac{d}{m} = 1/4$ and $\frac{e}{n} = 1/8$ as in Lemma 4.1, we recover the (improved) constant $c_0 = 10.24$, as opposed to the previous value $c_0 = 128$.

6 Improved Soundness for V_{RS} and V_{VRS}

We prove a much tighter soundness analysis for the verifiers V_{RS} and V_{VRS}, which respectively are supposed to test proximity to RS and VRS over subspaces of finite field extensions of $\mathbb{F}_2$. We show the following theorem:

Theorem 6.1. *The (strong) PCPP verifiers V_{RS} and V_{VRS} have respective soundness functions*

$$s_{\mathrm{RS}}(\delta, n) \geq \frac{1}{1 + \frac{2^{\eta+1}+\kappa^{\log c}}{1-2^{-\eta}}}\delta \quad \textit{and}$$
$$s_{\mathrm{VRS}}(\delta, n) \geq \frac{1}{2 + \frac{2^{\eta+1}+\kappa^{\log c}}{1-2^{-\eta}}}\delta \ ,$$

where $n = 2^\kappa$, $(\eta, \kappa_0, \gamma, \mu)$ are any integers satisfying

$$\begin{matrix} \min\left\{ \left\lfloor \frac{\kappa_0+1}{2} \right\rfloor - \gamma \, , \left\lceil \frac{\kappa_0+1}{2} \right\rceil + \gamma \right\} \geq 1 \\ \max\left\{ \left\lfloor \frac{\kappa_0+1}{2} \right\rfloor - \gamma + \mu + 1 \, , \left\lceil \frac{\kappa_0+1}{2} \right\rceil + \gamma \right\} < \kappa_0 + 1 \end{matrix} \quad \textit{and} \quad \mu + 1 \geq \eta \geq \max\{2, \mu+1\} \ ,$$

and

$$c \stackrel{\mathrm{def}}{=} \left(a - \frac{\sqrt{b(b+4a)} - b}{2} \right)^{\frac{1}{\min\left\{ \log\left(\frac{\kappa_0+1}{\lfloor (\kappa_0+1)/2 \rfloor - \gamma + \mu + 1} \right), \log\left(\frac{\kappa_0+1}{\lceil (\kappa_0+1)/2 \rceil + \gamma} \right) \right\}}} \ , \ \textit{where} \ \begin{cases} a \stackrel{\mathrm{def}}{=} \frac{2^{\eta+1}(1+c_0)}{2^{\eta-1}-1} \\ b \stackrel{\mathrm{def}}{=} 2^{\eta+1} \\ c_0 \stackrel{\mathrm{def}}{=} \frac{4}{\left(1 - \frac{1}{2^{\eta-1}} - \frac{1}{2^\eta}\right)^2} \end{cases} .$$

(Alternatively, for any given problem size $n = 2^\kappa$, one can use the method discussed in Section 6.1.3 to compute an "effective" c, which is usually much better than the theoretical expression given above.)

We perform our improved soundness analyses "from bottom to top"

- in Section 6.1 we analyze the soundness of $V_{\mathrm{RS},=}$;
- in Section 6.2 we analyze the soundness of $V_{\mathrm{RS},<}$;
- in Section 6.3 we analyze the soundness of $V_{\mathrm{RS},>}$;
- in Section 6.4 we analyze the soundness of V_{RS}; and
- in Section 6.5 we analyze the soundness of V_{RS}.

Remark 6.2 (From Strong PCPPs to "Weak" PCPPs)**.** The PCPPs obtained in Theorem 6.1 are of the "strong" kind, i.e., follow Definition 3.15, instead of the weaker Definition 3.14.

In general, (naive) sequential repetition of a strong PCPP verifier with soundness function $s\colon (0,1] \times \mathbb{N} \to (0,1]$ to obtain a ("weak") PCPP verifier with proximity parameter $\delta \in [0,1]$ and soundness parameter $s' \in [0,1]$ requires a number of repetitions $m_{s,\delta,s'}(n)$ such that

$$m_{s,\delta,s'}(n) = \left\lceil \frac{\log(1-s')}{\log\left(1 - s(\delta, n)\right)} \right\rceil \ , \tag{6}$$

where n is the length of the explicit input.

We are not interested in randomness-efficient sequential repetition (cf. [BSS08, Proposition 2.9]), because we are not concerned with saving randomness and naive sequential repetition is

much more efficient computationally. Hence, we shall always perform naive sequential repetition for the purpose of soundness amplification.

In particular, for the specific case $s' \stackrel{\text{def}}{=} 1/2$, we get that

$$m_{s,\delta,\frac{1}{2}}(n) = \left\lceil \frac{-1}{\log\left(1 - s(\delta, n)\right)} \right\rceil \leq \left\lceil \frac{1}{s(\delta, n)} \right\rceil \quad .$$

Thus, we can bound the number of repetitions of V_{RS} and V_{VRS} required for soundness $s'_{\text{RS}} := s'_{\text{VRS}} := 1/2$ and proximity parameters δ_{RS} and δ_{VRS} respectively as follows:

$$\left\lceil \frac{\log(1 - s'_{\text{RS}})}{\log(1 - s_{\text{RS}}(\delta_{\text{RS}}, 2^\kappa)}) \right\rceil \leq \left\lceil \frac{1}{s_{\text{RS}}(\delta_{\text{RS}}, 2^\kappa)} \right\rceil \leq \left\lceil \frac{1}{\delta_{\text{RS}}} \left(1 + \frac{2^{\eta+1} + \kappa^{\log c}}{1 - 2^{-\eta}}\right) \right\rceil$$

and

$$\left\lceil \frac{\log(1 - s'_{\text{VRS}})}{\log(1 - s_{\text{VRS}}(\delta_{\text{VRS}}, 2^\kappa)}) \right\rceil \leq \left\lceil \frac{1}{s_{\text{VRS}}(\delta_{\text{VRS}}, 2^\kappa)} \right\rceil \leq \left\lceil \frac{1}{\delta_{\text{VRS}}} \left(1 + \frac{2^{\eta+1} + \kappa^{\log c}}{1 - 2^{-\eta}}\right) \right\rceil \quad .$$

In general, we denote by

$$V_{\text{aRS}} \quad \text{and} \quad V_{\text{aVRS}}$$

the amplified versions of V_{RS} and V_{VRS}: they each take the same inputs as the non-amplified version, as well as the two additional inputs respectively specifying the target soundness and proximity parameter.

6.1 Soundness Analysis of $V_{\text{RS},=}$

Recall that the "engine" for testing proximity to RS is the verifier $V_{\text{RS},=}$, whose job is to test proximity to $\text{RS}\big(\mathbb{F}_{2^\ell}, L, |L|/8 - 1\big)$ where L is some κ-dimensional subspace of $\mathbb{F}_{2^\ell}$.

As mentioned in the introduction, we have generalized the construction of $V_{\text{RS},=}$ (its code is given in Section 10.3.1) and we shall provide a much higher soundness analysis, provided by the following theorem:

Theorem 6.3. *Consider a construction of $V_{\text{RS},=}$ parametrized by integers $(\eta, \kappa_0, \gamma, \mu)$ as described above, with*

$$\begin{matrix} \min\left\{ \left\lfloor \frac{\kappa_0+1}{2} \right\rfloor - \gamma \,,\, \left\lceil \frac{\kappa_0+1}{2} \right\rceil + \gamma \right\} \geq 1 \\ \max\left\{ \left\lfloor \frac{\kappa_0+1}{2} \right\rfloor - \gamma + \mu + 1 \,,\, \left\lceil \frac{\kappa_0+1}{2} \right\rceil + \gamma \right\} < \kappa_0 + 1 \end{matrix} \quad \text{and} \quad \mu + 1 \geq \eta \geq \max\{2, \mu + 1\} \quad . \tag{7}$$

There exists a constant $c \geq 1$ such that for every positive integer κ and every positive ε the following holds: if for a function $p\colon L \to \mathbb{F}_{2^\ell}$, a string π, an integer ℓ, and a κ-dimensional linear subset $L \subseteq \mathbb{F}_{2^\ell}$ it holds that

$$\Pr\left[V^{(p,\pi)}_{\text{RS},=}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big) = 1\right] > 1 - \varepsilon \tag{8}$$

then

$$\Delta_L\Big(p, \text{RS}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big)\Big) \leq c^{\log \kappa} \cdot \varepsilon \quad . \tag{9}$$

In fact, we can take c to be the following value:

$$c \stackrel{\text{def}}{=} \left(a - \frac{\sqrt{b(b+4a)} - b}{2}\right)^{\frac{1}{\min\left\{\log\left(\frac{\kappa_0+1}{\lfloor(\kappa_0+1)/2\rfloor-\gamma+\mu+1}\right), \log\left(\frac{\kappa_0+1}{\lceil(\kappa_0+1)/2\rceil+\gamma}\right)\right\}}} \quad , \text{ where } \begin{cases} a \stackrel{\text{def}}{=} \frac{2^{\eta+1}(1+c_0)}{2^{\eta-1}-1} > 0 \\ b \stackrel{\text{def}}{=} 2^{\mu+2} > 0 \\ c_0 \stackrel{\text{def}}{=} \frac{4}{\left(1-\frac{1}{2^{\eta-1}}-\frac{1}{2^{\eta}}\right)^2} \end{cases} .$$

(In particular, for any given $\eta \geq 2$ and $\kappa_0 \geq 1$, we are forced to choose $\mu = \eta - 1$ and we can optimally choose γ as the integer, among those allowed by Equation 7, minimizing the exponent in the above expression.)

The above theorem simply says that $V_{\mathrm{RS},=}$ is a (strong) PCPP verifier with inverse-polylogarithmic soundness:

Corollary 6.4. *The (strong) PCPP verifier $V_{\mathrm{RS},=}$ has soundness function*

$$s_{\mathrm{RS},=}(\delta, n) \geq \frac{\delta}{\kappa^{\log c}} .$$

We first prove two simple lemmas (Section 6.1.1), then provide the theoretical analysis of the soundness function (Section 6.1.2), and after that we explain how our analysis lets us numerically compute a better lower bound on the soundness for any given dimension (Section 6.1.3).

6.1.1 Two simple lemmas

We begin by proving two lemmas that are needed for the soundness analysis of $V_{\mathrm{RS},=}$ in Section 6.1.2. Roughly, the lemmas state that the relative distance of a function to bivariate polynomials whose degree is restricted in only one of the two variables can be understood as the "average" of the distance of the restriction of the function to a column to univariate low-degree polynomials:

Lemma 6.5. *Let $\mathbb{F}$ be a finite field, A and B subsets of $\mathbb{F}$, $g\colon A \times B \to \mathbb{F}$ a function, and d a positive integer. Then:*

$$\delta^{(d,*)}_{A\times B}(g) = \mathop{\mathbf{E}}_{b\in B}\left[\delta^{(d)}_{A}(g|^{\leftrightarrow}_{b})\right] . \tag{10}$$

Lemma 6.6. *Let $\mathbb{F}$ be a finite field, A and B subsets of $\mathbb{F}$, $g\colon A \times B \to \mathbb{F}$ a function, and e a positive integer. Then:*

$$\delta^{(*,e)}_{A\times B}(g) = \mathop{\mathbf{E}}_{a\in A}\left[\delta^{(e)}_{B}(g|^{\updownarrow}_{a})\right] . \tag{11}$$

The proofs of these lemmas were left implicit in [BSS08]. For completeness, we give the proof of Lemma 6.5; an analogous argument can be carried out to prove the (symmetric) Lemma 6.6.

Proof of Lemma 6.5. Re-writing the left-hand side of Equation 10:

$$\begin{aligned} \delta^{(d,*)}_{A\times B}(g) &= \min_{\substack{Q\in\mathbb{F}[x,y] \\ \deg_x(Q)\leq d}} \Delta_{A\times B}(g, Q) \\ &= \min_{\substack{Q\in\mathbb{F}[x,y] \\ \deg_x(Q)\leq d}} \frac{\left|\left\{(a,b) \in A\times B \,:\, g(a,b) \neq Q(a,b)\right\}\right|}{|A|\cdot|B|} \end{aligned}$$

$$= \frac{1}{|A| \cdot |B|} \cdot \min_{\substack{Q \in \mathbb{F}[x,y] \\ \deg_x(Q) \le d}} \sum_{b \in B} \left| \{ a \in A \,:\, g(a,b) \neq Q(a,b) \} \right| \,. \tag{12}$$

Re-writing the right-hand side of Equation 10:

$$\begin{aligned}
\mathop{\mathbf{E}}_{b \in B} \left[\delta_A^{(d)}(g|_b^{\leftrightarrow}) \right] &= \frac{1}{|B|} \cdot \sum_{b \in B} \delta_A^{(d)}(g|_b^{\leftrightarrow}) \\
&= \frac{1}{|B|} \cdot \sum_{b \in B} \min_{\substack{Q \in \mathbb{F}[x] \\ \deg_x(Q) \le d}} \Delta_A(g|_b^{\leftrightarrow}, Q) \\
&= \frac{1}{|B|} \cdot \sum_{b \in B} \min_{\substack{Q \in \mathbb{F}[x] \\ \deg_x(Q) \le d}} \frac{\left| \{ a \in A \,:\, g(a,b) \neq Q(a) \} \right|}{|A|} \\
&= \frac{1}{|A| \cdot |B|} \cdot \sum_{b \in B} \min_{\substack{Q \in \mathbb{F}[x] \\ \deg_x(Q) \le d}} \left| \{ a \in A \,:\, g(a,b) \neq Q(a) \} \right| \,.
\end{aligned} \tag{13}$$

Let P be a bivariate polynomial over $\mathbb{F}$ (with degree in x at most d) that minimizes the summation in Equation 12; hence,

$$\delta_{A \times B}^{(d,*)}(g) = \frac{1}{|A| \cdot |B|} \cdot \sum_{b \in B} \left| \{ a \in A \,:\, g(a,b) \neq P(a,b) \} \right| \,.$$

For every $b \in B$: define the univariate polynomial P_b over $\mathbb{F}$ by $P_b(x) := P(x,b)$; note that the degree of P_b is at most d; observe that

$$\begin{aligned}
\left| \{ a \in A \,:\, g(a,b) \neq P(a,b) \} \right| &= \left| \{ a \in A \,:\, g(a,b) \neq P_b(x) \} \right| \\
&\ge \min_{\substack{Q \in \mathbb{F}[x] \\ \deg_x(Q) \le d}} \left| \{ a \in A \,:\, g(a,b) \neq Q(a) \} \right| \,.
\end{aligned}$$

Thus,

$$\begin{aligned}
\delta_{A \times B}^{(d,*)}(g) &= \frac{1}{|A| \cdot |B|} \cdot \sum_{b \in B} \left| \{ a \in A \,:\, g(a,b) \neq P(a,b) \} \right| \\
&\ge \frac{1}{|A| \cdot |B|} \cdot \sum_{b \in B} \min_{\substack{Q \in \mathbb{F}[x] \\ \deg_x(Q) \le d}} \left| \{ a \in A \,:\, g(a,b) \neq Q(a) \} \right| \\
&= \mathop{\mathbf{E}}_{b \in B} \left[\delta_A^{(d)}(g|_b^{\leftrightarrow}) \right] \,.
\end{aligned} \tag{14}$$

We now prove the other inequality, necessary to deduce equality. For every $b \in B$: let R_b be the univariate polynomial over $\mathbb{F}$ (with degree in x at most d) that minimizes the b-summand in the summation of Equation 13, so that

$$\mathop{\mathbf{E}}_{b \in B} \left[\delta_A^{(d)}(g|_b^{\leftrightarrow}) \right] = \frac{1}{|A| \cdot |B|} \cdot \sum_{b \in B} \left| \{ a \in A \,:\, g(a,b) \neq R_b(a) \} \right| \,.$$

Letting, for every $b \in B$, S_b denote any univariate polynomial over $\mathbb{F}$ such that $S_b(b) = 1$ and $S_b(B - \{b\}) = \{0\}$, define the bivariate polynomial P over $\mathbb{F}$ by $P(x, y) := \sum_{b \in B} S_b(y) R_b(x)$; note that the degree of P in x is at most d. Observe that

$$\begin{aligned}\sum_{b \in B} \left| \left\{ a \in A \,:\, g(a,b) \neq R_b(a) \right\} \right| &= \sum_{b \in B} \left| \left\{ a \in A \,:\, g(a,b) \neq S_b(b) R_b(a) \right\} \right| \\ &= \sum_{b \in B} \left| \left\{ a \in A \,:\, g(a,b) \neq P(a,b) \right\} \right| \\ &\geq \min_{\substack{Q \in \mathbb{F}[x,y] \\ \deg_x(Q) \leq d}} \sum_{b \in B} \left| \left\{ a \in A \,:\, g(a,b) \neq Q(a,b) \right\} \right| \,.\end{aligned}$$

Thus,

$$\begin{aligned}\mathop{\mathbf{E}}_{b \in B} \left[\delta_A^{(d)}(g|_b^{\leftrightarrow}) \right] &= \frac{1}{|A| \cdot |B|} \cdot \sum_{b \in B} \left| \left\{ a \in A \,:\, g(a,b) \neq R_b(a) \right\} \right| \\ &\geq \frac{1}{|A| \cdot |B|} \cdot \min_{\substack{Q \in \mathbb{F}[x,y] \\ \deg_x(Q) \leq d}} \sum_{b \in B} \left| \left\{ a \in A \,:\, g(a,b) \neq Q(a,b) \right\} \right| \\ &= \delta_{A \times B}^{(d,*)}(g) \,. \end{aligned} \tag{15}$$

From Equation 14 and Equation 15, we deduce Equation 10, as desired. □

6.1.2 The proof by induction

We proceed to the proof of Theorem 6.3. Our proof follows the approach of Ben-Sasson and Sudan, which is described at high level in [BSS08, Section 6.1]; in fact, we make an explicit effort to structure our proof in the same way as was done by Ben-Sasson and Sudan, so as to benefit from their intuitive discussions (of both the algorithm and the proof), which will apply here too. For the construction of the generalized $V_{\mathrm{RS},=}$, see Section 10.3.1; throughout, we assume basic familiarity with $V_{\mathrm{RS},=}$.

In short, the new parameters are as follows: we consider testing the generic degree $|L|/2^\eta - 1$ (instead of $|L|/8 - 1$), with the base case at a generic dimension κ_0 (instead of 6), a linear subspace L_0 with γ less dimensions, and a linear subspace L_0' with μ more dimensions than L_0.

Without loss of generality, we assume that ε is chosen so that $\varepsilon < c^{-\log \kappa}$, for otherwise there is nothing to prove because the relative distance Δ_L is at most 1.

Proof of Theorem 6.3. The proof is by induction on κ, the dimension of L.

The base case $\kappa \leq \kappa_0$ is immediate: by construction, $V_{\mathrm{RS},=}$ accepts the implicit input $p \colon L \to \mathbb{F}_{2^\ell}$ if and only if $p \in \mathrm{RS}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big)$, because $V_{\mathrm{RS},=}$ queries all the values of p, and then verifies that the polynomial corresponding to p (obtained via interpolation) has degree at most $|L|/2^\eta - 1$.

If instead $\kappa > \kappa_0$, assume that the lemma is true for all $\kappa' \in \mathbb{N}$ less than κ. Our goal is now to prove that the lemma holds for κ as well. So assume that the oracle pair (p, π) is accepted with probability greater than $1 - \varepsilon$ (Equation 8); we want to show that there exists a polynomial P of degree at most $|L|/2^\eta - 1$ such that its evaluation over L is at distance at most $D(\kappa) \cdot \varepsilon$ from p (Equation 9), and want the smallest c such that $D(\kappa) \leq c^{\log \kappa}$.

Step 0: basic notation. Let $(a_1, \ldots, a_\kappa)$ be any basis for the linear subset L. Define the linear subsets L_0, L'_0, L_1, L_β of L as follows:

$$\begin{aligned}
L_0 &:= (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma}) \ , \text{ so that } \dim(L_0) = \lfloor \tfrac{\kappa}{2} \rfloor - \gamma \ ; \\
L'_0 &:= (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}) \ , \text{ so that } \dim(L'_0) = \lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu \ ; \\
\forall\, \beta \in L_1,\ L_\beta &:= \begin{cases} \operatorname{span}(L'_0, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1}) & \beta \in L'_0 \\ \operatorname{span}(L'_0, \beta) & \beta \notin L'_0 \end{cases} \ , \text{ so that } \dim(L_\beta) = \lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1 \ ; \\
L_1 &:= (a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_\kappa) \ , \text{ so that } \dim(L_1) = \lceil \tfrac{\kappa}{2} \rceil + \gamma \ .
\end{aligned}$$

Note that L_0, L'_0, L_1, L_β are defined following [BSS08, Definition 6.4], with the exception of the additional parameters γ and μ that we introduce to control their sizes. Essentially, γ lets us choose a different decomposition of L into L_0 and L_1 (originally, $\gamma = 0$), and μ lets us control how much bigger is L'_0 as compared to L_0 (originally, $\mu = 2$). Also, Equation 7 guarantees that L_0, L'_0, L_1, L_β all have positive dimension and all have dimension *less* than κ (that is, at most κ to make sense, and in fact strictly less than it so we make "progress").

Next define the linearized polynomial q to be the vanishing polynomial of L_0 (as in [BSS08, Section 6.2]), and the proximity proof $\pi = (f, \Pi)$ as in [BSS08, Definition 6.4]. Finally, recall the definition of the two disjoint subsets T and S of $\mathbb{F}_{2^\ell} \times \mathbb{F}_{2^\ell}$,

$$T := \bigcup_{\beta \in L_1} \Big((L_0 + \beta) \times \{q(\beta)\} \Big) \quad \text{and} \quad S := \left(\bigcup_{\beta \in L_1} \big(L_\beta \times \{q(\beta)\} \big) \right) - T \ ,$$

and the definition of the two (partial) bivariate functions $\hat{p} \colon T \to \mathbb{F}_{2^\ell}$ and $\hat{f} \colon S \cup T \to \mathbb{F}_{2^\ell}$,

$$\hat{p}(\alpha, \beta') := p(\alpha) \quad \text{and} \quad \hat{f}(\alpha, \beta') := \begin{cases} f(\alpha, \beta') & \text{if } (\alpha, \beta') \in S \\ \hat{p}(\alpha, \beta') & \text{if } (\alpha, \beta') \in T \end{cases} \ .$$

Finally, for a subset W of $\mathbb{F}_{2^\ell} \times \mathbb{F}_{2^\ell}$, we call the β-row of W the set $W_\beta = \{\alpha : (\alpha, \beta) \in W\}$ and the α-column of W the set $W_\alpha = \{\beta : (\alpha, \beta) \in W\}$; given a function $g \colon W \to \mathbb{F}_{2^\ell} \times \mathbb{F}_{2^\ell}$ the restriction of g to the β-row of W is denoted $g|^{\leftrightarrow}_{\beta}$ and the restriction of g to the α-column of W is denoted by $g|^{\updownarrow}_{\alpha}$.

Given the above notation, recall that $V^{(p,\pi)}_{\mathrm{RS},=}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big)$ will, with equal probability, do one of the following two tests:

- "row test": choose a random $\beta \in L_1$ and recurse on $V^{(\hat{f}|^{\leftrightarrow}_{q(\beta)}, \pi^{\leftrightarrow}_{q(\beta)})}_{\mathrm{RS},=}\big(\mathbb{F}_{2^\ell}, L_\beta, |L_\beta|/2^\eta - 1\big)$, or
- "column test": choose a random $\alpha \in L'_0$ and recurse on $V^{(\hat{f}|^{\updownarrow}_{\alpha}, \pi^{\updownarrow}_{\alpha})}_{\mathrm{RS},=}\big(\mathbb{F}_{2^\ell}, L'_0, |L'_0|/2^\eta - 1\big)$.

Step 1: restricting the bivariate function $\hat{f}$ to a product set $L'_0 \times q(L_1)$. Define the following quantities:

- for every $\beta \in L_1$, $\varepsilon(\beta)$ is the probability that the inner verifier rejects $(\hat{f}|^{\leftrightarrow}_{q(\beta)}, \pi^{\leftrightarrow}_{q(\beta)})$;
- for every $\alpha \in L'_0$, $\varepsilon(\alpha)$ is the probability that the inner verifier rejects $(\hat{f}|^{\updownarrow}_{\alpha}, \pi^{\updownarrow}_{\alpha})$;
- $\varepsilon_{\mathsf{row}} := \mathbf{E}_{\beta \in L_1}[\varepsilon(\beta)]$;

- $\varepsilon_{\mathsf{col}} := \mathbf{E}_{\alpha \in L_0'}[\varepsilon(\alpha)]$;
- $d := |L_\beta|/2^\eta - 1$ (d is the same regardless of the choice of $\beta \in L_1$);
- $e := |q(L_1)|/2^\eta - 1$.

Note that, by construction of $V_{\mathrm{RS},=}$, $\varepsilon = (\varepsilon_{\mathsf{row}} + \varepsilon_{\mathsf{col}})/2$ (as a random row test or a random column test is performed, with equal probability); in particular, $\varepsilon_{\mathsf{row}}, \varepsilon_{\mathsf{col}} \leq 2\varepsilon$. Also, by [BSS08, Proposition 6.5], we deduce that:

- for every $\beta \in L_1$, $\hat{f}|^{\leftrightarrow}_{q(\beta)}$ has domain $L_\beta \times \{q(\beta)\}$;
- for every $\alpha \in L_0'$, $\hat{f}|^{\updownarrow}_{\alpha}$ has domain $\{\alpha\} \times q(L_1)$.

Using the inductive assumption, we deduce that:

$$\mathop{\mathbf{E}}_{\beta \in L_1}\left[\delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{q(\beta)})\right] \leq \mathop{\mathbf{E}}_{\beta \in L_1}\left[D(\dim L_\beta)\cdot\varepsilon(\beta)\right] = D(\lfloor\tfrac{\kappa}{2}\rfloor - \gamma + \mu + 1)\cdot\varepsilon_{\mathsf{row}} \quad \text{and}$$
$$\mathop{\mathbf{E}}_{\alpha \in L_0'}\left[\delta^{(e)}_{q(L_1)}(\hat{f}|^{\updownarrow}_{\alpha})\right] \leq \mathop{\mathbf{E}}_{\alpha \in L_0'}\left[D(\dim q(L_1))\cdot\varepsilon(\alpha)\right] = D(\lceil\tfrac{\kappa}{2}\rceil + \gamma)\cdot\varepsilon_{\mathsf{col}} \ .$$

Define the function $f'\colon L_0' \times q(L_1) \to \mathbb{F}_{2^\ell}$ by $f' := \hat{f}|_{L_0' \times q(L_1)}$, i.e., f' is the restriction of $\hat{f}$ to the product set $L_0' \times q(L_1)$. Then:

- Using Lemma 6.5, as well as the facts that $L_0' \subseteq L_\beta$ and $|L_0'| = |L_\beta|/2$, we deduce that
$$\delta^{(d,*)}_{L_0' \times q(L_1)}(f') = \mathop{\mathbf{E}}_{\beta \in L_1}\left[\delta^{(d)}_{L_0'}(f'|^{\leftrightarrow}_{q(\beta)})\right] \leq \mathop{\mathbf{E}}_{\beta \in L_1}\left[2\cdot\delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{q(\beta)})\right] \leq 2\cdot D(\lfloor\tfrac{\kappa}{2}\rfloor - \gamma + \mu + 1)\cdot\varepsilon_{\mathsf{row}} \ .$$
- Using Lemma 6.6, as well as the fact that $f'|^{\updownarrow}_{\alpha} = \hat{f}|^{\updownarrow}_{\alpha}$, we deduce that
$$\delta^{(*,e)}_{L_0' \times q(L_1)}(f') = \mathop{\mathbf{E}}_{\alpha \in L_0'}\left[\delta^{(e)}_{q(L_1)}(f'|^{\updownarrow}_{\alpha})\right] = \mathop{\mathbf{E}}_{\alpha \in L_0'}\left[\delta^{(e)}_{q(L_1)}(\hat{f}|^{\updownarrow}_{\alpha})\right] \leq D(\lceil\tfrac{\kappa}{2}\rceil + \gamma)\cdot\varepsilon_{\mathsf{col}} \ .$$

Observing that

$$d = |L_\beta|/2^\eta - 1 \leq |L_0'|/2^{\eta-1} \quad \text{and} \quad e = |q(L_1)|/2^\eta - 1 \leq |q(L_1)|/2^\eta \ ,$$

we can now invoke Theorem 4.2 with $m = |L_0'|$, $n = |q(L_1)|$, $A = L_0'$, $B = q(L_1)$, $g = f'$, and the d and e of this proof (indeed, because $\eta \geq 2$ from Equation 7, we get that $\frac{d}{m} + \frac{e}{n} \leq \frac{1}{4} + \frac{1}{2} < 1$, so that the hypothesis of the theorem is satisfied), we get:

$$\begin{aligned}\delta^{(d,e)}_{L_0' \times q(L_1)}(f') \leq& c_0 \cdot \left(\delta^{(d,*)}_{L_0' \times q(L_1)}(f') + \delta^{(*,e)}_{L_0' \times q(L_1)}(f')\right) \\ \leq& c_0 \cdot \left(2\cdot D(\lfloor\tfrac{\kappa}{2}\rfloor - \gamma + \mu + 1)\cdot\varepsilon_{\mathsf{row}} + D(\lceil\tfrac{\kappa}{2}\rceil + \gamma)\cdot\varepsilon_{\mathsf{col}}\right) \ ,\end{aligned}$$

where

$$c_0 = \max\left\{3, \frac{4}{\left(1 - \frac{1}{2^{\eta-1}} - \frac{1}{2^\eta}\right)^2}\right\} = \frac{4}{\left(1 - \frac{1}{2^{\eta-1}} - \frac{1}{2^\eta}\right)^2} \ .$$

Step 2: Extending the analysis to the bivariate function $\hat{p}$. We make the following definitions:

- let $Q(x,y)$ be the polynomial over $\mathbb{F}_{2^\ell}$ corresponding to a codeword in $\mathrm{RM}(\mathbb{F}_{2^\ell}, L_0' \times q(L_1), (d,e))$ closest to f'; in particular, $\delta^{(d,e)}_{L_0' \times q(L_1)}(f') = \Delta_{L_0' \times q(L_1)}(f', Q)$;
- for every $\beta \in L_1$, define the function $\hat{f}|^{\leftrightarrow'}_{q(\beta)} \colon L_0' \to \mathbb{F}_{2^\ell}$ by $\hat{f}|^{\leftrightarrow'}_{q(\beta)} := (\hat{f}|^{\leftrightarrow}_{q(\beta)})|_{L_0'}$ (and recall that L_β is the domain of $\hat{f}|^{\leftrightarrow}_{q(\beta)}$, $L_0' \subseteq L_\beta$, and $|L_0'| = |L_\beta|/2$); and
- for every $\beta \in L_1$, let $Q_{q(\beta)}(x)$ be the polynomial over $\mathbb{F}_{2^\ell}$ corresponding to a codeword in $\mathrm{RS}(\mathbb{F}_{2^\ell}, L_\beta, d)$ closest to $\hat{f}|^{\leftrightarrow}_{q(\beta)}$; in particular, $\delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{q(\beta)}) = \Delta_{L_\beta}(\hat{f}|^{\leftrightarrow}_{q(\beta)}, Q_{q(\beta)})$.

For every $\beta' \in q(L_1)$, we say that β' is *good* if $Q_{\beta'}(x) = Q(x,\beta')$; otherwise we say that β' is *bad*. Define $T_{\mathsf{good}} := \{(\alpha,\beta') \in T \,:\, \beta' \text{ is good}\}$.

We now bound the probability that, over a random $(\alpha,\beta') \in T$, $\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta')$:

$$\begin{aligned}
\Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta')\Big] &= \Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\Big|\, \beta' \text{ is bad}\Big] \cdot \Pr_{\beta'\in q(L_1)}\Big[\beta' \text{ is bad}\Big] \\
&\quad + \Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\Big|\, \beta' \text{ is good}\Big] \cdot \Pr_{\beta'\in q(L_1)}\Big[\beta' \text{ is good}\Big] \\
&\leq 1 \cdot \Pr_{\beta'\in q(L_1)}\Big[\beta' \text{ is bad}\Big] \\
&\quad + \Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\Big|\, \beta' \text{ is good}\Big] \cdot 1 \\
&= \Pr_{\beta'\in q(L_1)}\Big[\beta' \text{ is bad}\Big] + \Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\Big|\, \beta' \text{ is good}\Big] \,.
\end{aligned} \tag{16}$$

We begin with a bound on the first summand Equation 16. Observe that, for every $\beta' \in q(L_1)$, if β' is bad, then $Q_{\beta'}(x) \neq Q(x,\beta')$; since the degrees of both $Q_{\beta'}(x)$ and $Q(x,\beta')$ are at most $d = |L_\beta|/2^\eta - 1 \leq |L_0'|/2^{\eta-1}$, then $Q_{\beta'}(x)$ and $Q(x,\beta')$ may agree on at most a $1/2^{\eta-1}$ fraction of the points in L_0'. Therefore, by the triangle inequality,

$$\frac{2^{\eta-1}-1}{2^{\eta-1}} \leq \Delta_{L_0'}\big(Q_{\beta'}(x), Q(x,\beta')\big) \leq \Delta_{L_0'}\big(Q_{\beta'}(x), \hat{f}|^{\leftrightarrow'}_{q(\beta)}\big) + \Delta_{L_0'}\big(\hat{f}|^{\leftrightarrow'}_{q(\beta)}, Q(x,\beta')\big) \,.$$

For reasons that will become clear shortly, we try to understand the expectation, taken over a random $\beta' \in q(L_1)$, of the above two distances.

From $L_0' \subseteq L_\beta$ and $|L_0'| = |L_\beta|/2$, we obtain that

$$\Delta_{L_0'}\big(Q_{q(\beta)}(x), \hat{f}|^{\leftrightarrow'}_{q(\beta)}\big) \leq 2 \cdot \Delta_{L_\beta}\big(Q_{q(\beta)}(x), \hat{f}|^{\leftrightarrow}_{q(\beta)}\big) = 2 \cdot \delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{q(\beta)}) \,,$$

so that

$$\mathop{\mathbf{E}}_{\beta\in L_1}\Big[\Delta_{L_0'}\big(Q_{q(\beta)}(x), \hat{f}|^{\leftrightarrow'}_{q(\beta)}\big)\Big] \leq 2 \cdot \mathop{\mathbf{E}}_{\beta\in L_1}\Big[\delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{q(\beta)})\Big] \,.$$

Next, observe that, for every $\beta' \in q(L_1)$, $\hat{f}|^{\leftrightarrow'}_{\beta'} = f'|^{\leftrightarrow}_{\beta'}$, because f' is simply the restriction of $\hat{f}$ to $L_0' \times q(L_1)$ and $\hat{f}|^{\leftrightarrow'}_{\beta}$ is the restriction of $\hat{f}|^{\leftrightarrow}_{\beta}$ to L_0'. Hence, for every $\beta' \in q(L_1)$: it holds that $\Delta_{L_0'}\big(\hat{f}|^{\leftrightarrow'}_{\beta'}, Q(x,\beta')\big) = \Delta_{L_0'}\big(f'|^{\leftrightarrow}_{\beta'}, Q(x,\beta')\big)$, so that

$$\mathop{\mathbf{E}}_{\beta'\in q(L_1)}\Big[\Delta_{L_0'}\big(\hat{f}|^{\leftrightarrow'}_{\beta'}, Q(x,\beta')\big)\Big] = \mathop{\mathbf{E}}_{\beta'\in q(L_1)}\Big[\Delta_{L_0'}\big(f'|^{\leftrightarrow}_{\beta'}, Q(x,\beta')\big)\Big]$$

$$
\begin{aligned}
&= \frac{1}{|q(L_1)|} \cdot \sum_{\beta' \in q(L_1)} \Delta_{L'_0}\big(f'|^{\leftrightarrow}_{\beta'}, Q(x,\beta')\big) \\
&= \frac{1}{|q(L_1)|} \cdot \sum_{\beta' \in q(L_1)} \frac{\Big|\big\{\alpha \in L'_0 \,:\, Q(\alpha,\beta') \neq f'|^{\leftrightarrow}_{\beta'}(\alpha)\big\}\Big|}{|L'_0|} \\
&= \frac{1}{|q(L_1)|} \cdot \sum_{\beta' \in q(L_1)} \frac{\Big|\big\{\alpha \in L'_0 \,:\, Q(\alpha,\beta') \neq f'(\alpha,\beta')\big\}\Big|}{|L'_0|} \\
&= \frac{\Big|\big\{(\alpha,\beta') \in L'_0 \times q(L_1) \,:\, Q(\alpha,\beta') \neq f'(\alpha,\beta')\big\}\Big|}{|L'_0| \cdot |q(L_1)|} \\
&= \Delta_{L'_0 \times q(L_1)}(f', Q) \\
&= \delta^{(d,e)}_{L'_0 \times q(L_1)}(f') \ .
\end{aligned}
$$

Hence, we get:

$$
\begin{aligned}
&\Pr_{\beta \in L_1}\Big[q(\beta) \text{ is bad}\Big] \\
&\leq \Pr_{\beta \in L_1}\left[\Delta_{L'_0}\big(Q_{q(\beta)}(x), \hat{f}|^{\leftrightarrow'}_{q(\beta)}\big) + \Delta_{L'_0}\big(\hat{f}|^{\leftrightarrow'}_{q(\beta)}, Q(x,q(\beta))\big) \geq \frac{2^{\eta-1}-1}{2^{\eta-1}}\right] \\
&\leq \frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \mathop{\mathbf{E}}_{\beta \in L_1}\left[\Delta_{L'_0}\big(Q_{q(\beta)}(x), \hat{f}|^{\leftrightarrow'}_{q(\beta)}\big) + \Delta_{L'_0}\big(\hat{f}|^{\leftrightarrow'}_{q(\beta)}, Q(x,q(\beta))\big)\right] \quad \text{(via Markov's inequality)} \\
&\leq \frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \mathop{\mathbf{E}}_{\beta \in L_1}\left[\Delta_{L'_0}\big(Q_{q(\beta)}(x), \hat{f}|^{\leftrightarrow'}_{q(\beta)}\big) + \Delta_{L'_0}\big(\hat{f}|^{\leftrightarrow'}_{q(\beta)}, Q(x,q(\beta))\big)\right] \\
&= \frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \left(\mathop{\mathbf{E}}_{\beta \in L_1}\left[\Delta_{L'_0}\big(Q_{q(\beta)}(x), \hat{f}|^{\leftrightarrow'}_{q(\beta)}\big)\right] + \mathop{\mathbf{E}}_{\beta \in L_1}\left[\Delta_{L'_0}\big(\hat{f}|^{\leftrightarrow'}_{q(\beta)}, Q(x,q(\beta))\big)\right]\right) \\
&\leq \frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \left(2 \cdot \mathop{\mathbf{E}}_{\beta \in L_1}\left[\delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{q(\beta)})\right] + \delta^{(d,e)}_{L'_0 \times q(L_1)}(f')\right) \\
&\leq \frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \left(2 \cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1) \cdot \varepsilon_{\mathsf{row}} + c_0 \cdot \left(2 \cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1) \cdot \varepsilon_{\mathsf{row}} + D(\lceil \tfrac{\kappa}{2} \rceil + \gamma) \cdot \varepsilon_{\mathsf{col}}\right)\right) \\
&= \frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \left((2+2c_0) \cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1) \cdot \varepsilon_{\mathsf{row}} + c_0 \cdot D(\lceil \tfrac{\kappa}{2} \rceil + \gamma) \cdot \varepsilon_{\mathsf{col}}\right) \\
&= \frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \left((2+2c_0) \cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1) \cdot \varepsilon_{\mathsf{row}} - c_0 \cdot D(\lceil \tfrac{\kappa}{2} \rceil + \gamma) \cdot \varepsilon_{\mathsf{row}} + 2 \cdot c_0 \cdot D(\lceil \tfrac{\kappa}{2} \rceil + \gamma) \cdot \varepsilon\right) \\
&\leq \frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \left\{ \begin{array}{ll} (2+2c_0) \cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1) \cdot 2\varepsilon & \text{if } \begin{array}{c}(2+2c_0) \cdot D(\lfloor \frac{\kappa}{2} \rfloor - \gamma + \mu + 1) \cdot \varepsilon_{\mathsf{row}} \\ \geq c_0 \cdot D(\lceil \frac{\kappa}{2} \rceil + \gamma) \cdot \varepsilon_{\mathsf{row}}\end{array} \\ 2 \cdot c_0 \cdot D(\lceil \tfrac{\kappa}{2} \rceil + \gamma) \cdot \varepsilon & \text{otherwise} \end{array} \right\} \\
&\leq \frac{2^{\eta}}{2^{\eta-1}-1} \cdot \max\left\{(2+2c_0) \cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1)\,,\, c_0 \cdot D(\lceil \tfrac{\kappa}{2} \rceil + \gamma)\right\} \cdot \varepsilon \ ,
\end{aligned}
$$

which is our upper bound for the first summand of Equation 16.

We now bound the second summand of Equation 16. First, recalling that the function $\hat{p}\colon T \to \mathbb{F}_{2^\ell}$ agrees with $\hat{f}\colon S \cup T \to \mathbb{F}_{2^\ell}$ on the set $T \subseteq S \cup T$, we get:

$$
\Pr_{(\alpha,\beta') \in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\Big|\, \beta' \text{ is good}\Big]
$$

$$
\begin{aligned}
&= \frac{\Pr_{(\alpha,\beta')\in T}\left[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \wedge \beta' \text{ is good}\right]}{\Pr_{(\alpha,\beta')\in T}\left[\beta' \text{ is good}\right]} \\
&= \frac{\left|\left\{(\alpha,\beta')\in T \,:\, \hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \wedge \beta' \text{ is good}\right\}\right|}{\left|\left\{(\alpha,\beta')\in T \,:\, \beta' \text{ is good}\right\}\right|} \\
&= \frac{\left|\left\{(\alpha,\beta')\in T \,:\, \hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \wedge \beta' \text{ is good}\right\}\right|}{|S\cup T|} \cdot \frac{|S\cup T|}{\left|\left\{(\alpha,\beta')\in T \,:\, \beta' \text{ is good}\right\}\right|} \\
&\leq \frac{\left|\left\{(\alpha,\beta')\in S\cup T \,:\, \hat{f}(\alpha,\beta') \neq Q(\alpha,\beta') \wedge \beta' \text{ is good}\right\}\right|}{|S\cup T|} \cdot \frac{|S\cup T|}{\left|\left\{(\alpha,\beta')\in T \,:\, \beta' \text{ is good}\right\}\right|} \\
&= \Pr_{(\alpha,\beta')\in S\cup T}\left[\hat{f}(\alpha,\beta') \neq Q(\alpha,\beta')\right] \cdot \frac{|S\cup T|}{|T_{\mathsf{good}}|} \\
&\leq \min\left\{\delta^{(d,*)}_{S\cup T}(\hat{f})\,,\, \delta^{(*,e)}_{S\cup T}(\hat{f})\right\} \cdot \frac{|S\cup T|}{|T_{\mathsf{good}}|} \\
&\leq \delta^{(d,*)}_{S\cup T}(\hat{f}) \cdot \frac{|S\cup T|}{|T_{\mathsf{good}}|} \\
&= \mathop{\mathbf{E}}_{\beta\in L_1}\left[\delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{q(\beta)})\right] \cdot \frac{|S\cup T|}{|T_{\mathsf{good}}|} \\
&\leq D(\lfloor\tfrac{\kappa}{2}\rfloor - \gamma + \mu + 1) \cdot \varepsilon_{\mathsf{row}} \cdot \frac{|S\cup T|}{|T_{\mathsf{good}}|} \\
&\leq 2\cdot D(\lfloor\tfrac{\kappa}{2}\rfloor - \gamma + \mu + 1) \cdot \varepsilon \cdot \frac{|S\cup T|}{|T_{\mathsf{good}}|}\ .
\end{aligned}
$$

For every $\beta \in L_1$:

- by [BSS08, Proposition 6.3], the $q(\beta)$-row of T is $L_0 + \beta$, an affine shift of L_0 by β; and
- by [BSS08, Proposition 6.5], the $q(\beta)$-row of $S \cup T$ is L_β;

hence, from $L_0 + \beta \subseteq L_\beta$ and $|L_\beta| = 2^{\mu+1}|L_0|$, we deduce that $|S\cup T|/|T_{\mathsf{good}}| = 2^{\mu+1}$. Therefore,

$$
\frac{|S\cup T|}{|T_{\mathsf{good}}|} = \frac{|S\cup T|}{|T|}\cdot\frac{|T|}{|T_{\mathsf{good}}|} = 2^{\mu+1}\cdot\frac{1}{\Pr_{\beta'\in q(L_1)}\left[\beta' \text{ is good}\right]} = 2^{\mu+1}\cdot\frac{1}{1-\Pr_{\beta'\in q(L_1)}\left[\beta' \text{ is bad}\right]}\ .
$$

Thus, the second summand of Equation 16 can be upper bounded as follows:

$$
\Pr_{(\alpha,\beta')\in T}\left[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\middle|\, \beta' \text{ is good}\right] \leq \frac{2^{\mu+2}\cdot D(\lfloor\frac{\kappa}{2}\rfloor - \gamma + \mu + 1)}{1-\Pr_{\beta'\in q(L_1)}\left[\beta' \text{ is bad}\right]}\cdot\varepsilon\ .
$$

In conclusion, we have established that:

$$
\Pr_{(\alpha,\beta')\in T}\left[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta')\right] \leq \Pr_{\beta'\in q(L_1)}\left[\beta' \text{ is bad}\right] + \frac{2^{\mu+2}\cdot D(\lfloor\frac{\kappa}{2}\rfloor - \gamma + \mu + 1)}{1-\Pr_{\beta'\in q(L_1)}\left[\beta' \text{ is bad}\right]}\cdot\varepsilon\ , \tag{17}
$$

where

$$\Pr_{\beta' \in q(L_1)}\left[\beta' \text{ is bad}\right] \leq \frac{2^{\eta}}{2^{\eta-1}-1} \cdot \max\{(2+2c_0)\cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1)\,,\, c_0 \cdot D(\lceil \tfrac{\kappa}{2} \rceil + \gamma)\} \cdot \varepsilon \ . \quad (18)$$

Now recall that we have assumed that $D(\kappa) \leq c^{\log \kappa}$; we now seek the smallest c for which we can show that

$$\Pr_{(\alpha,\beta') \in T}\left[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta')\right] \leq c^{\log \kappa} \cdot \varepsilon \ . \quad (19)$$

We observe that

$$D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1) \leq c^{\log(\lfloor \frac{\kappa}{2} \rfloor - \gamma + \mu + 1)} \leq \frac{c^{\log \kappa}}{c^{\log\left(\frac{\kappa_0+1}{\lfloor (\kappa_0+1)/2 \rfloor - \gamma + \mu + 1}\right)}} \quad (20)$$

and

$$D(\lceil \tfrac{\kappa}{2} \rceil + \gamma) \leq c^{\log(\lceil \frac{\kappa}{2} \rceil + \gamma)} \leq \frac{c^{\log \kappa}}{c^{\log\left(\frac{\kappa_0+1}{\lceil (\kappa_0+1)/2 \rceil + \gamma}\right)}} \ . \quad (21)$$

Next, combining Equation 17 and Equation 18, by substituting the upper bounds from Equation 20 and Equation 21, and recalling that by assumption $c^{\log \kappa} \cdot \varepsilon \leq 1$, we get:

$$\begin{aligned}
&\Pr_{(\alpha,\beta') \in T}\left[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta')\right] \\
&\leq \left(\frac{2^{\eta}}{2^{\eta-1}-1} \cdot \max\{(2+2c_0)\cdot c^{-\log\left(\frac{\kappa_0+1}{\lfloor (\kappa_0+1)/2 \rfloor - \gamma + \mu + 1}\right)}\,,\, c_0 \cdot c^{-\log\left(\frac{\kappa_0+1}{\lceil (\kappa_0+1)/2 \rceil + \gamma}\right)}\} \right. \\
&\left. + \frac{2^{\mu+2} \cdot c^{-\log\left(\frac{\kappa_0+1}{\lfloor (\kappa_0+1)/2 \rfloor - \gamma + \mu + 1}\right)}}{1 - \frac{2^{\eta}}{2^{\eta-1}-1} \cdot \max\{(2+2c_0)\cdot c^{-\log\left(\frac{\kappa_0+1}{\lfloor (\kappa_0+1)/2 \rfloor - \gamma + \mu + 1}\right)}\,,\, c_0 \cdot c^{-\log\left(\frac{\kappa_0+1}{\lceil (\kappa_0+1)/2 \rceil + \gamma}\right)}\}} \right) \cdot c^{\log \kappa} \cdot \varepsilon \ ,
\end{aligned} \quad (22)$$

So, noting that

$$\begin{aligned}
&\max\{(2+2c_0)\cdot c^{-\log\left(\frac{\kappa_0+1}{\lfloor (\kappa_0+1)/2 \rfloor - \gamma + \mu + 1}\right)}\,,\, c_0 \cdot c^{-\log\left(\frac{\kappa_0+1}{\lceil (\kappa_0+1)/2 \rceil + \gamma}\right)}\} \\
&\qquad \leq (2+2c_0) c^{\max\left\{-\log\left(\frac{\kappa_0+1}{\lfloor (\kappa_0+1)/2 \rfloor - \gamma + \mu + 1}\right), -\log\left(\frac{\kappa_0+1}{\lceil (\kappa_0+1)/2 \rceil + \gamma}\right)\right\}} \\
&\qquad\qquad = (2+2c_0) c^{-\min\left\{\log\left(\frac{\kappa_0+1}{\lfloor (\kappa_0+1)/2 \rfloor - \gamma + \mu + 1}\right), \log\left(\frac{\kappa_0+1}{\lceil (\kappa_0+1)/2 \rceil + \gamma}\right)\right\}} \ ,
\end{aligned}$$

it suffices to require that

$$\frac{a}{x(c)} + \frac{b \cdot \frac{1}{x(c)}}{1 - \frac{a}{x(c)}} \leq 1 \ , \text{ where } \begin{cases} a \stackrel{\text{def}}{=} \dfrac{2^{\eta+1}(1+c_0)}{2^{\eta-1}-1} > 0 \\ b \stackrel{\text{def}}{=} 2^{\mu+2} > 0 \\ x(c) \stackrel{\text{def}}{=} c^{\min\left\{\log\left(\frac{\kappa_0+1}{\lfloor (\kappa_0+1)/2 \rfloor - \gamma + \mu + 1}\right), \log\left(\frac{\kappa_0+1}{\lceil (\kappa_0+1)/2 \rceil + \gamma}\right)\right\}} > 0 \end{cases} . \quad (23)$$

Solving the (second-degree) inequality in $x(c)$, we obtain that the smaller among the two solutions is

$$a > x(c) \geq a - \frac{\sqrt{b(b+4a)} - b}{2} \ . \quad (24)$$

Moreover, $a - \frac{\sqrt{b(b+4a)}-b}{2} > 0$ always holds (as can be verified via the definitions of a and b). Therefore, we choose c so that $x(c) = a - \frac{\sqrt{b(b+4a)}-b}{2}$. In other words, we choose c such that

$$c \stackrel{\text{def}}{=} \left(a - \frac{\sqrt{b(b+4a)} - b}{2} \right)^{\frac{1}{\min\left\{\log\left(\frac{\kappa_0+1}{\lfloor(\kappa_0+1)/2\rfloor-\gamma+\mu+1}\right), \log\left(\frac{\kappa_0+1}{\lceil(\kappa_0+1)/2\rceil+\gamma}\right)\right\}}}, \tag{25}$$

and this choice of c will ensure that Equation 19 is satisfied.

Step 3: From bivariate $\hat{p}$ to univariate p. Let $P(x)$ be the polynomial over $\mathbb{F}_{2^\ell}$ defined by $P(x) := Q(x, q(x))$; its degree is at most $|L|/2^\eta - 1$, as is possible to see by direct computation:

$$\begin{aligned}
\deg P &= \deg_x(Q) + \deg q \cdot \deg_y(Q) \\
&\leq d + |L_0| \cdot e \\
&= \left(\frac{|L_\beta|}{2^\eta} - 1\right) + |L_0| \cdot \left(\frac{|q(L_1)|}{2^\eta} - 1\right) \\
&= \left(\frac{|L_\beta|}{2^\eta} - 1\right) + |L_0| \cdot \left(\frac{|L_1|}{2^\eta} - 1\right) \\
&= \left(2^{\dim(L_\beta)-\eta} - 1\right) + 2^{\dim(L_0)} \cdot \left(2^{\dim(L_1)-\eta} - 1\right) \\
&= \left(2^{\lfloor\kappa/2\rfloor-\gamma+\mu+1-\eta} - 1\right) + 2^{\lfloor\kappa/2\rfloor-\gamma} \cdot \left(2^{\lceil\kappa/2\rceil+\gamma-\eta} - 1\right) \\
&= (2^{\kappa-\eta} - 1) + (2^{\lfloor\kappa/2\rfloor-\gamma+\mu+1-\eta} - 2^{\lfloor\kappa/2\rfloor-\gamma}) \\
&= (2^{\kappa-\eta} - 1) + 2^{\lfloor\kappa/2\rfloor-\gamma} \cdot (2^{\mu+1-\eta} - 1) \qquad \text{(from Equation 7, } \mu+1 \leq \eta\text{)} \\
&\leq (2^{\kappa-\eta} - 1) + 0 \\
&= \frac{|L|}{2^\eta} - 1 \ .
\end{aligned}$$

Thus, the evaluation of P on L is a codeword in $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big)$, and, to finish the proof, it suffices to show that the fractional distance of p to the evaluation of P on L is at most $c^{\log\kappa} \cdot \varepsilon$.

And, indeed, from the upperbound provided by Equation 19 and from the definition of T as the set $T = \{(\tau, q(\tau)) : \tau \in L\}$, we deduce that for all but at most a $(c^{\log\kappa} \cdot \varepsilon)$-fraction of the elements in L it holds that

$$p(\tau) = \hat{p}(\tau, q(\tau)) = Q(\tau, q(\tau)) = P(\tau),$$

meaning that p and the evaluation of P on L do agree on all but at most a $(c^{\log\kappa} \cdot \varepsilon)$-fraction of the elements in L, as desired. □

Remark 6.7 (Completeness)**.** In the above proof, we have not used the constraint $\mu + 1 \geq \eta$ from Equation 7. This constraint in fact comes from the *completeness* proof, which can be easily carried out in our more generic case by following the proof (in the special case) of [BSS08, Proposition 6.9]. We need to show that:

> If $p\colon L \to \mathbb{F}_{2^\ell}$ is the evaluation of a polynomial $P(x)$ of degree less than $|L|/2^\eta$, then there exists a proximity proof π for which $V_{\mathrm{RS},=}^{(p,\pi)}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big)$ always accepts.

We argue by induction, constructing a (partial) bivariate function $f\colon S \to \mathbb{F}_{2^\ell}$ for which the function $\hat{f}\colon S \cup T \to \mathbb{F}_{2^\ell}$ is such that the $q(\beta)$-row of $\hat{f}$ is a codeword of $\mathrm{RS}(\mathbb{F}_{2^\ell}, L_\beta, |L_\beta|/2^\eta - 1)$ and the α-column of $\hat{f}$ is a codeword of $\mathrm{RS}(\mathbb{F}_{2^\ell}, q(L_1), |q(L_1)|/2^\eta - 1)$ for every $\beta \in L_1$ and $\alpha \in L_0'$.

Let $Q(x, y) := P(x) \bmod y - q(x)$, where q is the vanishing polynomial of L_0, so that:

- $P(x) = Q(x, q(x))$, and
- $\deg_x(Q) < |L_0|$ and $\deg_y(Q) = \lfloor \frac{\deg(P)}{\deg(q)} \rfloor < (|L|/2^\eta)/|L_0| = |L_1|/2^\eta = |q(L_1)|/2^\eta$.

We can then set $f(\alpha, \beta') = Q(\alpha, \beta')$ for every $(\alpha, \beta') \in S$. Then, observe that $\hat{f}$ is the evaluation of Q on $S \cup T$: indeed, whenever $(\alpha, \beta') \in S$ this follows from $\hat{f}(\alpha, \beta') = f(\alpha, \beta') = Q(\alpha, \beta')$; and, whenever $(\alpha, \beta') \in T$, we have that $\beta' = q(\alpha)$, so that

$$\hat{p}(\alpha, \beta') = \hat{p}(\alpha, q(\alpha)) = p(\alpha) = P(\alpha) = Q(\alpha, q(\alpha)) = Q(\alpha, \beta') \enspace .$$

Therefore:

- for each $\beta \in L_1$, the $q(\beta)$-row of $S \cup T$ is L_β and, by construction, the restriction of $\hat{f}$ to L_β is the evaluation of $Q(x, q(\beta))$ over L_β; hence, because $\mu + 1 \geq \eta$,
$$\deg(Q(x, q(\beta))) \leq \deg_x(Q) < |L_0| = \frac{|L_\beta|}{2^{\mu+1}} \leq \frac{|L_\beta|}{2^\eta} \enspace .$$
- for each $\alpha \in L_0'$, the α-column of $\hat{f}$ is $q(L_1)$ and, by construction, the restriction of $\hat{f}$ to $q(L_1)$ is the evaluation of $Q(\alpha, y)$ on $q(L_1)$; hence,
$$\deg(Q(\alpha, y)) \leq \deg_y(Q) < \frac{|q(L_1)|}{2^\eta} \enspace .$$

This completes the proof of completeness.

6.1.3 Soundness for concrete dimensions

While the soundness analysis of Section 6.1.2 provides vast improvements in the asymptotic constant c in the exponent of κ in the soundness lower bound $s_{\mathrm{RS},=}(\delta, n) \geq \frac{\delta}{\kappa^{\log c}}$, we seek more. This time, however, we shall turn to a concrete analysis, and only worry for soundness of all practical problem sizes — our asymptotic analysis was set up in a way that it could easily also yield a concrete soundness analysis.

Our starting point is Equation 17 and Equation 18 that we can summarize as the following requirement on $D(\kappa)$:

$$D(\kappa) = \begin{cases} \text{if } \kappa \leq \kappa_0 \ : & 1 \\ \text{if } \kappa > \kappa_0 \ : & \frac{2^\eta}{2^{\eta-1}-1} \cdot \max\{(2+2c_0) \cdot D(\lfloor \frac{\kappa}{2} \rfloor - \gamma + \mu + 1)\,,\, c_0 \cdot D(\lceil \frac{\kappa}{2} \rceil + \gamma)\} \\ & + \frac{2^{\mu+2} \cdot D(\lfloor \frac{\kappa}{2} \rfloor - \gamma + \mu + 1)}{1 - \frac{2^\eta}{2^{\eta-1}-1} \cdot \max\{(2+2c_0) \cdot D(\lfloor \frac{\kappa}{2} \rfloor - \gamma + \mu + 1)\,,\, c_0 \cdot D(\lceil \frac{\kappa}{2} \rceil + \gamma)\}} \end{cases} \enspace , \quad (26)$$

The above recursive function lets us, for any given dimension κ, compute the "effective" $\log c$ for κ as follows:

$$\log c(\kappa) \stackrel{\text{def}}{=} \frac{\log D(\kappa)}{\log \kappa} \enspace . \quad (27)$$

The behavior of $\log c(\kappa)$ is pleasantly well below the asymptotic bound for all practical sizes (e.g., for κ up to 100 or 200).

6.2 Soundness Analysis of $V_{\mathrm{RS},<}$

The (strong) PCPP verifier $V_{\mathrm{RS},<}$ tests proximity to $\mathrm{RS}(\mathbb{F}, S, d)$ for $d < |S|/2^{\eta} - 1$; it is constructed using the (strong) PCPP verifier $V_{\mathrm{RS},=}$; see Section 10.3.2.

The soundness of $V_{\mathrm{RS},<}$ was already analyzed (in the special case $\eta = 3$) in [BSS08, Proposition 6.13]. We give here a tighter analysis of the soundness of $V_{\mathrm{RS},<}$, where, compared to [BSS08, Proposition 6.13], (beyond keeping track of the parameter η) we improve on the reduction from the soundness of $V_{\mathrm{RS},=}$ by almost a factor of 2 in the distance argument of the soundness function.

Lemma 6.8. *If $V_{\mathrm{RS},=}$ has monotone soundness function $s_{\mathrm{RS},=}(\delta, n)$, then $V_{\mathrm{RS},<}$ has soundness function*

$$s_{\mathrm{RS},<}(\delta, n) \geq s_{\mathrm{RS},=}\left(\frac{2^{\eta}-1}{2^{\eta+1}}\delta, n\right) \quad .$$

Proof. Fix an explicit input $(\mathbb{F}, S, d)$ and an implicit input $p\colon S \to \mathbb{F}$ to $V_{\mathrm{RS},<}$ such that $d < d_{\kappa,\eta}$, where $d_{\kappa,\eta} := |S|/2^{\eta} - 1$. Also let $\pi = (\pi_1, \pi_2)$ be any proximity proof given to $V_{\mathrm{RS},<}$. Define $\delta := \Delta_S\big(p, \mathrm{RS}(\mathbb{F}, S, d)\big)$ and assume that $\delta > 0$.

Consider a parameter $\mu \in (0, 1)$ that we will optimize later. We distinguish between two cases:

- *Case 1:* $\Delta_S\big(p, \mathrm{RS}(\mathbb{F}, S, d_{\kappa,\eta})\big) > \mu\delta$.

  Then, by the soundness of $V_{\mathrm{RS},=}$ and the monotonicity of $s_{\mathrm{RS},=}$, the first subtest of $V_{\mathrm{RS},<}^{(p,\pi)}$, which is $V_{\mathrm{RS},=}^{(p,\pi_1)}(\mathbb{F}, S)$, rejects with probability at least $s_{\mathrm{RS},=}(\mu\delta, n)$.

- *Case 2:* $\Delta_S\big(p, \mathrm{RS}(\mathbb{F}, S, d_{\kappa,\eta})\big) \leq \mu\delta$.

  Then, p is both δ-far from $\mathrm{RS}(\mathbb{F}, S, d)$ and $\mu\delta$-close to $\mathrm{RS}(\mathbb{F}, S, d_{\kappa,\eta})$. Therefore, there exists a polynomial $P\colon \mathbb{F} \to \mathbb{F}$ such that $d < \deg(P) \leq d_{\kappa,\eta}$ and whose evaluation table over S is $\mu\delta$-close to p. Define the polynomial $Q\colon \mathbb{F} \to \mathbb{F}$ by $Q(x) := x^{d_{\kappa,\eta}-d}$, let q be its evaluation table over S, and define the function $p'\colon S \to \mathbb{F}$ by $p' = q \cdot p$. Then, p' is $\mu\delta$-close to the evaluation over S of the polynomial $P'\colon \mathbb{F} \to \mathbb{F}$ defined by $P'(x) := Q(x) \cdot P(x)$; note that $\deg(P') \leq (d_{\kappa,\eta} - d) + d = d_{\kappa,\eta}$. Since $d_{\kappa,\eta} = |S|/2^{\eta} - 1$, the evaluation table of P' over S is $\frac{2^{\eta}-1}{2^{\eta}}$-far from $\mathrm{RS}(\mathbb{F}, S, d_{\kappa,\eta})$, so that p' is $\big(\frac{2^{\eta}-1}{2^{\eta}} - \mu\delta\big)$-far from $\mathrm{RS}(\mathbb{F}, S, d_{\kappa,\eta})$. Hence, by the soundness of $V_{\mathrm{RS},=}$, the second subtest of $V_{\mathrm{RS},<}^{(p,\pi)}$, which is $V_{\mathrm{RS},=}^{(p',\pi_2)}(\mathbb{F}, S)$, rejects with probability at least $s_{\mathrm{RS},=}\big(\frac{2^{\eta}-1}{2^{\eta}} - \mu\delta, n\big)$, which, by the monotonicity of $s_{\mathrm{RS},=}$, is at least $s_{\mathrm{RS},=}\big(\frac{2^{\eta}-1}{2^{\eta}}\delta - \mu\delta, n\big)$.

Letting $\mu = \frac{2^{\eta}-1}{2^{\eta+1}}$ (the solution to $\mu = \frac{2^{\eta}-1}{2^{\eta}} - \mu$) completes the proof of the lemma. □

Corollary 6.9. *The (strong) PCPP verifier $V_{\mathrm{RS},<}$ has soundness function*

$$s_{\mathrm{RS},<}(\delta, n) \geq \frac{2^{\eta}-1}{2^{\eta+1}} \cdot \frac{\delta}{\kappa^{\log c}} \quad .$$

Proof. Immediate from Corollary 6.4 and Lemma 6.8. □

6.3 Soundness Analysis of $V_{\mathrm{RS},>}$

The (strong) PCPP verifier $V_{\mathrm{RS},>}$ tests proximity to $\mathrm{RS}(\mathbb{F}, S, d)$ for $d > |S|/2^\eta - 1$; it is constructed using the (strong) PCPP verifiers $V_{\mathrm{RS},=}$ and $V_{\mathrm{RS},<}$; see Section 10.3.3.

The soundness of $V_{\mathrm{RS},>}$ was already analyzed (in the special case $\eta = 3$) in [BSS08, Proposition 6.13]. We give here a tighter analysis of the soundness of $V_{\mathrm{RS},>}$, where, compared to [BSS08, Proposition 6.13], (beyond keeping track of the parameter η) we improve on the reduction from the soundness of $V_{\mathrm{RS},=}$ and $V_{\mathrm{RS},<}$ by optimizing over how the case analysis is split.

Lemma 6.10. *If $V_{\mathrm{RS},=}$ and $V_{\mathrm{RS},<}$ have monotone soundness functions $s_{\mathrm{RS},=}(\delta, n)$ and $s_{\mathrm{RS},<}(\delta, n)$, then, for any $\tau \in (0,1)$, $V_{\mathrm{RS},>}$ has soundness function*

$$s_{\mathrm{RS},>}(\delta, n) \geq \min\left\{(1-\tau)\delta\,,\; s_{\mathrm{RS},=}\left(\frac{\tau}{2^\eta}\delta, n\right)\,,\; s_{\mathrm{RS},<}\left(\frac{\tau}{2^\eta}\delta, n\right)\right\}\ .$$

Proof. Fix an explicit input $(\mathbb{F}, S, d)$ and an implicit input $p\colon S \to \mathbb{F}$ to $V_{\mathrm{RS},>}$ such that $d > d_{\kappa,\eta}$, where $d_{\kappa,\eta} := |S|/2^\eta - 1$. Without loss of generality, $d < |S| = 2^\eta(d_{\kappa,\eta}+1)$, otherwise p is always in $\mathrm{RS}(\mathbb{F}, S, d)$. Also let $\pi = \big((p_0, \pi_0), \ldots, (p_{2^\eta-1}, \pi_{2^\eta-1})\big)$ be any proximity proof given to $V_{\mathrm{RS},>}$. Define $\delta := \Delta_S\big(p, \mathrm{RS}(\mathbb{F}, S, d)\big)$ and assume that $\delta > 0$.

Define the function $q\colon S \to \mathbb{F}$ by $q(\alpha) = \sum_{i=0}^{2^\eta-1} \alpha^{i(d_{\kappa,\eta}+1)} p_i(\alpha)$ for all $\alpha \in S$. For any $\tau \in (0,1)$, we distinguish between two cases:

- *Case 1:* $\Delta_S(p, q) > (1-\tau)\delta$.

  Then, by construction, the second subtest of $V_{\mathrm{RS},>}$ rejects with probability at least $(1-\tau)\delta$.

- *Case 2:* $\Delta_S(p, q) \leq (1-\tau)\delta$.

  Then, there exists some p_i such that p_i is $\frac{\tau\delta}{2^\eta}$-far from $\mathrm{RS}(\mathbb{F}, S, d_i)$. If $d_i = d_{\kappa,\eta}$, then the first subtest of $V_{\mathrm{RS},>}^{(p,\pi)}$, which includes the test $V_{\mathrm{RS},=}^{(p_i,\pi_i)}(\mathbb{F}, S)$, rejects with probability at least $s_{\mathrm{RS},=}\big(\frac{\tau\delta}{2^\eta}, n\big)$; if instead $d_i < d_{\kappa,\eta}$, then the first subtest of $V_{\mathrm{RS},>}^{(p,\pi)}$, which includes the test $V_{\mathrm{RS},<}^{(p_i,\pi_i)}(\mathbb{F}, S, d_i)$, rejects with probability at least $s_{\mathrm{RS},<}\big(\frac{\tau\delta}{2^\eta}, n\big)$.

Thus the soundness is given by the minimum among the three possible rejection probabilities. □

Corollary 6.11. *The (strong) PCPP verifier $V_{\mathrm{RS},>}$ has soundness function*

$$s_{\mathrm{RS},>}(\delta, n) \geq \frac{1}{1 + \frac{2^{\eta+1} + \kappa^{\log c}}{1-2^{-\eta}}}\delta\ .$$

Proof. From Lemma 6.8, we know that $s_{\mathrm{RS},<}(\delta, n) \leq s_{\mathrm{RS},=}(\delta, n)$, so, from Lemma 6.10, we know that $s_{\mathrm{RS},>}(\delta, n) \geq \min\{(1-\tau)\delta\,,\, s_{\mathrm{RS},<}(\frac{\tau}{2^\eta}\delta, n)\}$. Moreover, from Corollary 6.9, we know that $s_{\mathrm{RS},<}(\frac{\tau}{2^\eta}\delta, n) \geq \frac{\tau}{2^\eta}\frac{2^\eta-1}{2^{\eta+1}}\frac{\delta}{\kappa^{\log c}}$.

Therefore, we need to (optimally) choose $\tau \in (0,1)$ to ensure that $(1-\tau)\delta = \frac{\tau}{2^\eta}\frac{2^\eta-1}{2^{\eta+1}}\frac{\delta}{\kappa^{\log c}}$; the equality is achieved for $\tau = \frac{1}{1+a}$, where $a = \frac{1-2^{-\eta}}{2^{\eta+1}k^{\log c}}$ which, once plugged back in $(1-\tau)\delta$ gives the claimed lower bound for $s_{\mathrm{RS},>}(\delta, n)$. □

6.4 Soundness Analysis of V_{RS}

The (strong) PCPP verifier V_{RS} tests proximity to $\mathrm{RS}(\mathbb{F}, S, d)$ with no restrictions on d; it is constructed using the (strong) PCPP verifiers $V_{\mathrm{RS},=}$, $V_{\mathrm{RS},<}$, and $V_{\mathrm{RS},>}$; see Section 10.3.4.

The soundness analysis of V_{RS} (implicit in [BSS08]) is trivial:

Lemma 6.12. *If $V_{\mathrm{RS},=}$, $V_{\mathrm{RS},<}$, and $V_{\mathrm{RS},>}$ have respective soundness functions $s_{\mathrm{RS},=}$, $s_{\mathrm{RS},<}$, and $s_{\mathrm{RS},>}$, then V_{RS} has soundness function*

$$s_{\mathrm{RS}}(\delta, n) \geq \min\left\{ s_{\mathrm{RS},=}(\delta, n)\,,\ s_{\mathrm{RS},<}(\delta, n)\,,\ s_{\mathrm{RS},>}(\delta, n) \right\}\ .$$

Proof. The verifier V_{RS} simply calls $V_{\mathrm{RS},<}$, $V_{\mathrm{RS},=}$, or $V_{\mathrm{RS},>}$, depending on whether the input degree d is less than, equal to, or greater than $d_{\kappa,\eta} := |S|/2^{\eta} - 1$. Hence, the soundness function of V_{RS} is no worse than the minimum among the soundness functions of $V_{\mathrm{RS},<}$, $V_{\mathrm{RS},=}$, and $V_{\mathrm{RS},>}$. □

Corollary 6.13. *The (strong) PCPP verifier V_{RS} has soundness function*

$$s_{\mathrm{RS}}(\delta, n) \geq \frac{1}{1 + \frac{2^{\eta+1} + \kappa^{\log c}}{1 - 2^{-\eta}}}\delta\ .$$

Proof. From Lemma 6.8 and Lemma 6.10 we know that $s_{\mathrm{RS},<}(\delta, n) \leq s_{\mathrm{RS},=}(\delta, n)$ and $s_{\mathrm{RS},>}(\delta, n) \leq s_{\mathrm{RS},<}(\delta, n)$ respectively. Hence, from Lemma 6.12, we know that $s_{\mathrm{RS}}(\delta, n) \geq s_{\mathrm{RS},>}(\delta, n)$, which can be lower bounded by Corollary 6.11, yielding the claimed lower bound. (And thus we have established the first lower bound from Theorem 6.1.) □

6.5 Soundness Analysis of V_{VRS}

The (strong) PCPP verifier V_{VRS} tests proximity to $\mathrm{VRS}(\mathbb{F}, S, H, d)$ with no restrictions on d; it is constructed using the (strong) PCPP verifier V_{RS}; see Section 10.3.5.

The soundness of V_{VRS} was already analyzed (in the special case $\eta = 3$) in [BSS08, Lemma 3.12]. We give here a tighter analysis of the soundness of the soundness of V_{VRS}, where, compared to [BSS08, Lemma 3.12], (beyond keeping track of the parameter η) we improve on the reduction from the soundness of V_{RS} by optimizing over how the case analysis is split.

Lemma 6.14. *If V_{RS} has monotone soundness function $s_{\mathrm{RS}}(\delta, n)$, then, for any $\tau \in (0, 1)$, V_{VRS} has soundness function*

$$s_{\mathrm{VRS}}(\delta, n) \geq \min\left\{ s_{\mathrm{RS}}(\tau\delta, n)\,,\ (1 - \tau)\delta \right\}\ .$$

Proof. Fix an explicit input $(\mathbb{F}, S, H, d)$ and an implicit input $p\colon S \to \mathbb{F}$ to V_{VRS}. Also let $\pi = (\widetilde{p}, \widetilde{\pi})$ be any proximity proof given to V_{VRS}. Define $\delta := \Delta_S\big(p, \mathrm{VRS}(\mathbb{F}, S, H, d)\big)$ and assume that $\delta > 0$.

We distinguish between two cases:

- *Case 1:* $\Delta_S\big(\widetilde{p}, \mathrm{RS}(\mathbb{F}, S, d - |H|)\big) > \tau\delta$.

  Then, by the soundness of V_{RS} and the monotonicity of s_{RS}, the first subtest of $V_{\mathrm{VRS}}^{(p,\pi)}$, which is $V_{\mathrm{RS}}^{(\widetilde{p},\widetilde{\pi})}(\mathbb{F}, S, d - |H|)$, rejects with probability at least $s_{\mathrm{RS}}(c\delta, n)$.

- *Case 2:* $\Delta_S\big(\widetilde{p}, \mathrm{RS}(\mathbb{F}, S, d-|H|)\big) \leq \tau\delta$.

  Then, there exists a polynomial $Q\colon \mathbb{F} \to \mathbb{F}$ of degree at most $d-|H|$ such that its evaluation table q over S is $\tau\delta$-close to $\widetilde{p}$. Let $Z_H\colon \mathbb{F} \to \mathbb{F}$ be the vanishing polynomial for the subspace H and let z_H be its evaluation over S. Observe that the function $z_H \cdot q$ is a codeword in $\mathrm{VRS}(\mathbb{F}, S, H, d)$; moreover, $\Delta_S(z_H \cdot \widetilde{p}, z_H \cdot q) \leq \tau\delta$. Therefore,

$$\begin{aligned}
\delta &= \Delta_S\big(p, \mathrm{VRS}(\mathbb{F}, S, H, d)\big) \\
&\leq \Delta_S\big(p, z_H \cdot \widetilde{p}\big) + \Delta_S\big(z_H \cdot \widetilde{p}, \mathrm{VRS}(\mathbb{F}, S, H, d)\big) \\
&= \Delta_S\big(p, z_H \cdot \widetilde{p}\big) + \Delta_S\big(z_H \cdot \widetilde{p}, z_H \cdot q\big) \\
&\leq \Delta_S\big(p, z_H \cdot \widetilde{p}\big) + \tau\delta \ ,
\end{aligned}$$

  so that p is at least $(1-\tau)\delta$-far from $z_H \cdot \widetilde{p}$, and thus the second subtest of $V_{\mathrm{VRS}}^{(p,\pi)}$ rejects with probability at least $(1-\tau)\delta$.

Thus the soundness is given by the minimum among the two possible rejection probabilities. □

Corollary 6.15. *The (strong) PCPP verifier V_{VRS} has soundness function*

$$s_{\mathrm{VRS}}(\delta, n) \geq \frac{1}{2 + \frac{2^{\eta+1}+\kappa^{\log c}}{1-2^{-\eta}}}\delta \ .$$

Proof. From Lemma 6.14, know that $s_{\mathrm{VRS}}(\delta, n) \geq \min\{s_{\mathrm{RS}}(\tau\delta, n)\,,\, (1-\tau)\delta\}$. Moreover, from Corollary 6.13, we know that $s_{\mathrm{RS}}(\delta, n) \geq \frac{1}{1+\frac{2^{\eta+1}+\kappa^{\log c}}{1-2^{-\eta}}}\delta$.

We need to (optimally) choose $\tau \in (0,1)$ to ensure that $(1-\tau)\delta = \tau\frac{1}{1+\frac{2^{\eta+1}+\kappa^{\log c}}{1-2^{-\eta}}}\delta$; the equality is achieved for $\tau = \frac{1}{1+a}$, where $a = \frac{1}{1+\frac{2^{\eta+1}+\kappa^{\log c}}{1-2^{-\eta}}}$ which, once plugged back in $(1-\tau)\delta$ gives the claimed lower bound for $s_{\mathrm{VRS}}(\delta, n)$. (And thus we have established the second lower bound from Theorem 6.1.) □

7 A Succinct Algebraic Constraint Satisfaction Problem

SuccinctACSP is a class of succinct algebraic constraint satisfaction problems, parametrized by a list of parameters, for univariate polynomials over finite field extensions of GF(2). Each particular SuccinctACSP problem simultaneously allows for the construction of PCPs with quasilinear prover and efficient verifier (as we will show in Section 8) and is flexible enough to support (several) fast reductions from high-level languages such as computations on random-access machines, as shown in [BSCGT12].

Informally, a choice par of parameters of SuccinctACSP consists of the following:

- A field size function $f\colon \mathbb{N} \to \mathbb{N}$, inducing a finite field family $\{\mathbb{F}_T\}_{T\in\mathbb{N}} := \{\mathrm{GF}(2^{f(T)})\}_{T\in\mathbb{N}}$.
- A family $\{H_T\}_{T\in\mathbb{N}}$ where each H_T is a subspace of $\mathbb{F}_T$.
- A family $\{N_T\}_{T\in\mathbb{N}}$, where each N_T is a set of affine functions.
- A family $\{P_T\}_{T\in\mathbb{N}}$, where each P_T is a constraint polynomial.
- A family $\{S_T\}_{T\in\mathbb{N}}$, where each S_T is a subspace of $\mathbb{F}_T$.

A reduction to SuccinctACSP will concretely instantiate a choice of the above parameters.

A pair (x, T) is a member in SuccinctACSP(par) if it fulfills the following: there exists a low-degree assignment polynomial $A\colon \mathbb{F}_T \to \mathbb{F}_T$ that "colors" elements of the field $\mathbb{F}_T$ such that (1) the constraint polynomial P_T at every element α of the subspace H_T, when given as input the colors in the "N_T-induced affine neighborhood" of α, is satisfied; and (2) the colors of the first $|x|$ elements of the subspace S_T are consistent with x.

Crucially, for each of the families in par, the T-th object must be able to be describable in time $\mathsf{polylog}(T)$. Additional properties that are essential for us to construct short PCPs with an efficient verifier include: for example, H_T is a subspace (so that one may leverage the computational properties of *linearized polynomials* [LN97, Section 2.5]), and functions in N_T are affine, that is, have degree equal to 1 (because any degree greater than 1 would cause the composition of a neighbor function with the assignment polynomial to have degree that is at least quadratic).

In the formal discussions, it will in fact be more convenient to index the above families by t, where the t-th elements will correspond to problems of size roughly 2^t.

Formally:

Definition 7.1 (Succinct Algebraic Constraint Satisfaction)**.** Consider the following parameters:

1. a field size function $f\colon \mathbb{N} \to \mathbb{N}$, inducing a family of finite fields $\{\mathbb{F}_t\}_{t\in\mathbb{N}}$ where $\mathbb{F}_t = \mathbb{F}_2(\mathsf{x})$ and x is the root of I_t, which is the irreducible polynomial of degree $f(t)$ over $\mathbb{F}_2$ output by FindIrrPoly$(1^{f(t)})$;

2. three functions associated with the family $\mathbf{H}$ in Parameter 3:
   (a) a dimension function $\mathsf{m}_{\mathbf{H}}\colon \mathbb{N} \to \mathbb{N}$,
   (b) a time function $\mathsf{t}_{\mathbf{H}}\colon \mathbb{N} \to \mathbb{N}$, and
   (c) a space function $\mathsf{s}_{\mathbf{H}}\colon \mathbb{N} \to \mathbb{N}$;

3. a family $\mathbf{H} = \{H_t\}_{t\in\mathbb{N}}$ such that:
   (a) H_t is a linear subset of $\mathbb{F}_t$,
   (b) $\dim(H_t) = \mathsf{m}_{\mathbf{H}}(t)$,
   (c) each linear subset H_t is specified by a basis $\left(\alpha_1^{\mathbf{H}}(\mathsf{x}), \ldots, \alpha_{\mathsf{m}_{\mathbf{H}}(t)}^{\mathbf{H}}(\mathsf{x})\right)$, and

(d) there exists a $\mathsf{t}_{\mathbf{H}}$-time $\mathsf{s}_{\mathbf{H}}$-space algorithm $\textsc{Find}\mathbf{H}$ such that $\textsc{Find}\mathbf{H}(1^t) = \left(\alpha_1^{\mathbf{H}}(\mathsf{x}), \ldots, \alpha_{\mathsf{m}_{\mathbf{H}}(t)}^{\mathbf{H}}(\mathsf{x})\right)$ for all $t \in \mathbb{N}$;

4. three functions associated with the family $\mathbf{N}$ in Parameter 5:
   (a) an affine neighborhood size function $\mathsf{c}_{\mathbf{N}} \colon \mathbb{N} \to \mathbb{N}$,
   (b) a time function $\mathsf{t}_{\mathbf{N}} \colon \mathbb{N} \to \mathbb{N}$, and
   (c) a space function $\mathsf{s}_{\mathbf{N}} \colon \mathbb{N} \to \mathbb{N}$;

5. a family $\mathbf{N} = \{N_t\}_{t \in \mathbb{N}}$ such that:
   (a) $N_t = (\mathsf{aff}_{t,i} \colon \mathbb{F}_t \to \mathbb{F}_t)_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)}$ is a sequence of $\mathsf{c}_{\mathbf{N}}(t)$ affine functions over $\mathbb{F}_t$,
   (b) each affine function $\mathsf{aff}_{t,i}$ is specified (in the straightforward way) by two elements $a_{t,i}(\mathsf{x})$ and $b_{t,i}(\mathsf{x})$ in $\mathbb{F}_t$, and
   (c) there exists a $\mathsf{t}_{\mathbf{N}}$-time $\mathsf{s}_{\mathbf{N}}$-space algorithm $\textsc{Find}\mathbf{N}$ such that $\textsc{Find}\mathbf{N}(1^t, i) = (a_{t,i}(\mathsf{x}), b_{t,i}(\mathsf{x}))$ for all $t \in \mathbb{N}$ and $i \in \{1, \ldots, \mathsf{c}_{\mathbf{N}}(t)\}$;

6. two functions associated with the family $\mathbf{D}$ in Parameter 7:
   (a) a time function $\mathsf{t}_{\mathbf{D}} \colon \mathbb{N} \to \mathbb{N}$, and
   (b) a space function $\mathsf{s}_{\mathbf{D}} \colon \mathbb{N} \to \mathbb{N}$;

7. a family $\mathbf{D} = \left\{(d_0^t, d_1^t, \ldots, d_{\mathsf{c}_{\mathbf{N}}(t)}^t)\right\}_{t \in \mathbb{N}}$ such that:
   (a) $d_i^t \in \mathbb{N}$ for $i = 0, 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$,
   (b) $d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1) \sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t \leq 2^{f(t)-2}$, and
   (c) there exists a $\mathsf{t}_{\mathbf{D}}$-time $\mathsf{s}_{\mathbf{D}}$-space algorithm $\textsc{Comp}\mathbf{D}$ such that $d_i^t = \textsc{Comp}\mathbf{D}(1^t, i)$ for every $t \in \mathbb{N}$ and $i \in \{0, 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)\}$;

8. four functions associated with the family $\mathbf{P}$ in Parameter 9:
   (a) a time function $\mathsf{t}_{\mathbf{P}} \colon \mathbb{N} \to \mathbb{N}$,
   (b) a space function $\mathsf{s}_{\mathbf{P}} \colon \mathbb{N} \to \mathbb{N}$,
   (c) a size function $\mathsf{S}_{\mathbf{P}} \colon \mathbb{N} \to \mathbb{N}$, and
   (d) a degree function $\mathsf{D}_{\mathbf{P}} \colon \mathbb{N} \to \mathbb{N}$;

9. a family $\mathbf{P} = \{P_t\}_{t \in \mathbb{N}}$ such that:
   (a) $P_t \colon \mathbb{F}_t^{1+\mathsf{c}_{\mathbf{N}}(t)} \to \mathbb{F}_t$ is a constraint polynomial,
   (b) there exists a $\mathsf{t}_{\mathbf{P}}$-time $\mathsf{s}_{\mathbf{P}}$-space algorithm $\textsc{Find}\mathbf{P}$ such that $[P_t]^{\mathsf{A}} = \textsc{Find}\mathbf{P}(1^t)$ is a $\mathsf{S}_{\mathbf{P}}$-size $\mathsf{D}_{\mathbf{P}}$-degree $\mathbb{F}_t$-algebraic circuit computing P_t for every $t \in \mathbb{N}$, and
   (c) $\mathsf{D}_{\mathbf{P}}(t)[i+1] \leq d_i^t$ for $i = 0, 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$;

10. two functions associated with the family $\mathbf{S}$ in Parameter 11:
    (a) a time function $\mathsf{t}_{\mathbf{S}} \colon \mathbb{N} \to \mathbb{N}$, and
    (b) a space function $\mathsf{s}_{\mathbf{S}} \colon \mathbb{N} \to \mathbb{N}$;

11. a family $\mathbf{S} = \{S_t\}_{t \in \mathbb{N}}$ such that:
    (a) S_t is a linear subset of H_t,
    (b) $\dim(S_t) = t$,
    (c) each linear subset S_t is specified by a basis $\left(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_t^{\mathbf{S}}(\mathsf{x})\right)$, and

(d) there exists a $\mathsf{t}_{\mathbf{S}}$-time $\mathsf{s}_{\mathbf{S}}$-space algorithm FIND$\mathbf{S}$ such that FIND$\mathbf{S}(1^t) = \big(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_t^{\mathbf{S}}(\mathsf{x})\big)$ for all $t \in \mathbb{N}$.

The language SUCCINCTACSP, with respect to a choice $\mathsf{par}_{\text{sACSP}}$ of the above parameters, consists of instances $(x, 1^t)$, where x is a binary string and t is an integer with $|x| \leq 2^t$, such that there exists an assignment polynomial $A\colon \mathbb{F}_t \to \mathbb{F}_t$ of degree less than $2^{\mathsf{m}_{\mathbf{H}}(t)}$ for which the following two conditions hold:

(i) *Satisfiability of constraints.* For every element $\alpha(\mathsf{x}) \in H_t$,

$$P_t\Big(\alpha(\mathsf{x}), \big(A \circ \mathsf{aff}_{t,1}\big)(\alpha(\mathsf{x})), \ldots, \big(A \circ \mathsf{aff}_{t,\mathsf{c}_{\mathbf{N}}(t)}\big)(\alpha(\mathsf{x}))\Big) = 0_{\mathbb{F}_t} \ .$$

If so, we say that the assignment polynomial A *satisfies the constraint polynomial* P_t.

(ii) *Consistency with the instance.* For every index $i \in \{1, \ldots, |x|\}$, letting $m := \lceil \log |x| \rceil$ and $\alpha_{i,m}(\mathsf{x})$ be the i-th element in $\mathrm{span}(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_m^{\mathbf{S}}(\mathsf{x}))$,

$$x = \mathrm{bit}\big(A(\alpha_{1,m}(\mathsf{x}))\big) \cdots \mathrm{bit}\big(A(\alpha_{|x|,m}(\mathsf{x}))\big) \ ,$$

where $\mathrm{bit}\colon \mathbb{F}_t \to \{0, 1, \perp\}$ maps $0_{\mathbb{F}_t}$ to 0, $1_{\mathbb{F}_t}$ to 1, and anything else to $\perp$. If so, we say that the assignment polynomial A *is consistent* with the instance $(x, 1^t)$.

Remark 7.2. Definition 7.1 follows related definitions that have appeared in [BSS08] and [BSGH+05]. We briefly discuss here how our definition compares to the previous ones.

The definition of Ben-Sasson and Sudan [BSS08, Definition 3.6] also considers low-degree polynomials vanishing at every affine neighborhood of a certain subset of a finite field; however, their definition (as is clear from its compactness!) does *not* require succinctness (for example, the subset H need not be linear) because their paper does not attempt to construct PCP verifiers that are efficient in time and space. (Note that, in the non-succinct case, the "consistency with the instance" requirement disappears.)

The later paper of Ben-Sasson et al. [BSGH+05], which constructs efficient PCP verifiers, does indeed give a definition [BSGH+05, Definition 6.1] for a succinct problem about polynomials (similar to the previous one of Ben-Sasson and Sudan [BSS08, Definition 3.6]), but is specific for the parameters obtained when reducing from bounded halting problems on Turing machines.

Definition 7.1 is thus a generalization of [BSGH+05, Definition 6.1] that leaves unspecified degrees of freedom that we believe important for "supporting" a variety of reductions from other models of computation.[8]

Remark 7.3. Ben-Sasson et al. [BSGH+05] also consider a family of problems about *multivariate* polynomials [BSGH+05, Definition 6.3]. We could have also considered a generalization to their definition, analogous to our Definition 7.1 that generalizes the univariate case [BSGH+05, Definition 6.1].

We choose to leave the investigation of such a definition (along with the possibility of finding even better reductions from natural problems "above" it, and better PCPs "below" it) for future work.

[8] Another technical difference is how the instance consistency check in SUCCINCTACSP is performed; in the case of Definition 7.1 this consistency check is taken to be of a very specific form, with the purpose of simplifying PCP constructions; [BSCGT12] do not use this degree of freedom, so this loss in generality is a minor one. (We could of course construct PCPs for a more general definition, but their additional complexity would not be justified.)

8 A PCP for Succinct ACSPs

We show how to use the PCPP verifiers V_{RS} and V_{VRS} discussed in Section 6, which respectively test proximity to RS and VRS over subspaces of finite field extensions of $\mathbb{F}_2$, to construct a PCP system $(P_{\text{ACSP}}, V_{\text{ACSP}})$ for the language SuccinctACSP (with respect to a given choice of its parameters) with quasilinear-time prover and succinct verifier (that only depends quasilinearly on the input instance).[9]

8.1 The Construction At High Level

Our construction follows and extends the one of Ben-Sasson and Sudan [BSS08]. Concretely, Ben-Sasson and Sudan showed how to test proximity to the Reed-Solomon code over additive subgroups of extension fields of $\mathbb{F}_2$ with quasilinear-size proximity proofs; using these, they then showed how to construct PCPs of quasilinear size for an algebraic constraint satisfaction problem that is a special case of SuccinctACSP, for the case where the verifier did not have to be succinct in time and without studying the prover complexity. Later, Ben-Sasson et al. [BSGH+05] developed techniques that allow the verifier in [BSS08] to run succinctly. (See Remark 8.9 for more details.)

As in [BSGH+05], we also need to ensure the succinctness of the verifier even if we ask the verifier to probabilistically check membership in SuccinctACSP(par), which involves probabilistically-checking properties of univariate polynomials with huge degrees.[10] Moreover, we also need to ensure that the verifier does not perform expensive computations on F, the "explicit" input at hand. Furthermore, in our case we also need to ensure that the prover *also* runs very fast.

We extend and improve [BSS08, BSGH+05] as follows:

- We use additive FFT techniques [Mat08] and computational properties of linearized polynomials to construct a generalization of the PCPP verifiers for Reed-Solomon codes of [BSS08] (together with a tighter soundness analysis that will allow us to show much better concrete efficiency, as was discussed in Section 2.3) where the prover runs in quasilinear time and the verifier runs succinctly.

- Given a Reed-Solomon proximity testing system such as the one discussed in the previous bullet, we show how to construct a PCP system for SuccinctACSP(par) where, roughly, the prover runs in $T\mathsf{poly}(T)$ time and the verifier runs in $|x|\mathsf{polylog}(T)$ time. The definition of SuccinctACSP will ensure that the verifier is able to succinctly generate appropriate representations for all the relevant objects (such as a small arithmetic circuit for the constraint polynomial P_T) as well as providing the appropriate additive subgroup structure necessary to leverage the proximity testing machinery for Reed-Solomon over sets with such a structure.

The construction is roughly as follows. Recall from Section 7 that in order to verify whether a given instance (x, T) is a member in SuccinctACSP(par), for a given choice of parameters par, one needs to establish whether there is a low-degree assignment polynomial A that satisfies two

[9] More precisely, we shall use the "amplified versions" V_{aRS} and V_{aVRS} of V_{RS} and V_{VRS}; see Remark 6.2.

[10] This, for example, is a difficulty that usually does not arise within algebraic PCPs using multivariate techniques; however such techniques are not known to yield quasilinear-size proofs (but only almost-linear [MR08]).

conditions: (i) when "composed" with the constraint polynomial P_T, A vanishes everywhere on H_T; and, moreover, (ii) A is "consistent" with x.

To check the first condition the PCP verifier V_{ACSP} works as follows. The verifier V_{ACSP} asks the prover to provide an evaluation p_0 of A along with a proximity proof π_0, so that he may test proximity of p_0 to the appropriate Reed-Solomon code. The verifier also asks the prover to provide an evaluation p_1 of B, the polynomial that one obtains when appropriately composing A with P_T, along with a proximity proof p_1 to the appropriate *Vanishing* Reed-Solomon code on the subspace H_T — that is the subcode consisting of low-degree polynomials that vanish over H_T. Testing proximity to the Vanishing Reed-Solomon code can be done if we already have a proximity tester for the Reed-Solomon code [BSS08, Lemma 3.12]. After this, the verifier performs a consistency check between p_0 and p_1 by verifying that the correct relation (i.e., as dictated by P_T) holds between p_0 and p_1.

To check the second condition V_{ACSP} works as follows. The verifier V_{ACSP} asks the prover to further provide a proximity proof π_{c} for the proximity of $p_0 - p_x$ to an appropriate Vanishing Reed-Solomon code on the subspace S_T, where p_x is a function depending on x that can be easily evaluated by the verifier. Intuitively, this will ensure that p_0 takes on the bits of x, as required, on the set S_T. (In our case, x will be the circuit description F which is given as input to the random-access machine.) The fact that consistency with the instance is done on a subspace is important to ensure that the prover and verifier can run in time quasilinear in the instance size.

Thus, the PCP prover P_{ACSP}, on input the witness polynomial A, will produce a PCP oracle π consisting of two pairs of strings (p_0, π_0) and (p_1, π_1) and a third string π_{c} such that:

- (p_0, π_0) is a pair consisting of an implicit input and a proximity proof attesting to the fact that A is of low-enough degree; concretely, p_0 is the evaluation table of A and π_0 is a proximity proof of p_0 to a Reed-Solomon code with appropriate parameters;
- (p_1, π_1) is a pair consisting of an implicit input and a proximity proof attesting to the fact that a polynomial B related to A is of low-enough degree and vanishes on an certain subset of a finite field; concretely, p_1 is the evaluation table of B and π_1 is a proximity proof of p_1 to a Vanishing Reed-Solomon code with appropriate parameters;
- π_{c} is a proximity proof attesting to the fact that $p_0 - p_x$, where p_x is a function depending on the input x, is of low-enough degree and vanishes on a certain subset of a finite field.

8.2 The Construction In Detail

Formally, we prove the following theorem:

Theorem 8.1 (PCPs for SuccinctACSP)**.** *Fix the language* SuccinctACSP *with a choice of parameters given by*

$$\Big(f, (\mathsf{m}_{\mathbf{H}}, \mathsf{t}_{\mathbf{H}}, \mathsf{s}_{\mathbf{H}}, \mathbf{H}), (\mathsf{c}_{\mathbf{N}}, \mathsf{t}_{\mathbf{N}}, \mathsf{s}_{\mathbf{N}}, \mathbf{N}), (\mathsf{t}_{\mathbf{D}}, \mathsf{s}_{\mathbf{D}}, \mathbf{D}), (\mathsf{t}_{\mathbf{P}}, \mathsf{s}_{\mathbf{P}}, \mathsf{S}_{\mathbf{P}}, \mathsf{D}_{\mathbf{P}}, \mathbf{P}), (\mathsf{t}_{\mathbf{S}}, \mathsf{s}_{\mathbf{S}}, \mathbf{S})\Big) \ .$$

There exists a PCP system $(P_{\text{ACSP}}, V_{\text{ACSP}})$ *for* SuccinctACSP *with perfect completeness and soundness* $1/2$*. Moreover,* P_{ACSP} *runs in quasilinear time and* V_{ACSP} *runs in polylogarithmic time (and both run quasilinearly in the instance size).*

We shall not conduct in this paper a detailed analysis of the prover time and space complexities and the verifier time and space complexities. While the algorithms that we present will fulfill the promise of a quasilinear-time prover and polylogarithmic-time verifier (with small exponents "in the polylog" and quasilinear running time in the input instance), we believe that a detailed analysis of these performance measures is best studied when accompanied with a code implementation.

Instead, in this paper, we shall concentrate on giving a detailed analysis of the query complexity, randomness complexity, and proof length; we do so in Appendix B.

We now give the construction of $(P_{\text{ACSP}}, V_{\text{ACSP}})$ in terms of V_{aRS} and V_{aVRS} (and the parameters of SuccinctACSP) and then prove its completeness and soundness properties.

Details for PCP system construction. We now give the details for the construction of the PCP system:

- in Construction 8.2 we describe the PCP prover P_{ACSP},
- in Definition 8.3 we define the "format" of the PCP oracle,
- in Construction 8.4 we describe the PCP prover V_{ACSP}.

The PCP prover P_{ACSP} for SuccinctACSP is as follows:

Construction 8.2 (PCP Prover for SuccinctACSP)**.** The PCP prover for SuccinctACSP, with respect to a choice

$$\Big(f, (\mathsf{m}_{\mathbf{H}}, \mathsf{t}_{\mathbf{H}}, \mathsf{s}_{\mathbf{H}}, \mathbf{H}), (\mathsf{c}_{\mathbf{N}}, \mathsf{t}_{\mathbf{N}}, \mathsf{s}_{\mathbf{N}}, \mathbf{N}), (\mathsf{t}_{\mathbf{D}}, \mathsf{s}_{\mathbf{D}}, \mathbf{D}), (\mathsf{t}_{\mathbf{P}}, \mathsf{s}_{\mathbf{P}}, \mathsf{S}_{\mathbf{P}}, \mathsf{D}_{\mathbf{P}}, \mathbf{P}), (\mathsf{t}_{\mathbf{S}}, \mathsf{s}_{\mathbf{S}}, \mathbf{S})\Big)$$

of parameters, is denoted P_{ACSP} and has hard-coded in it this choice of parameters. On input an instance $(x, 1^t)$ and an assignment polynomial $A\colon \mathbb{F}_t \to \mathbb{F}_t$ that is a witness to the membership of $(x, 1^t)$ in SuccinctACSP (see Definition 7.1), the PCP prover P_{ACSP} performs the following steps:

1. *Parameter instantiation:*
   (a) Generate the field extension $\mathbb{F}_t$ of $\mathbb{F}_2$: compute the degree $f(t)$, then compute the irreducible polynomial $I_t := \textsc{FindIrrPoly}\big(1^{f(t)}\big)$ with a root x, and let $\mathbb{F}_t := \mathbb{F}_2(\mathsf{x})$. (See Parameter 1 in Definition 7.1.) Note that a basis for $\mathbb{F}_t$ is given by $\mathcal{B}_{\mathbb{F}_t} := \big(1, \mathsf{x}, \ldots, \mathsf{x}^{f(t)-1}\big)$.
   (b) Find the basis for the linear subset H_t of $\mathbb{F}_t$ by computing $\mathcal{B}_{H_t} := \big(b_1^{(H_t)}, \ldots, b_{\mathsf{m}_{\mathbf{H}}(t)}^{(H_t)}\big) = \textsc{Find}\mathbf{H}(1^t)$. (See Parameter 3 in Definition 7.1.)
   (c) Find the affine functions $\{\mathsf{aff}_{t,i}(x) = a_{t,i} \cdot x + b_{t,i}\}_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)}$ that induce the "affine neighborhood" of each element in $\mathbb{F}_t$: for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, compute $(a_{t,i}, b_{t,i}) := \textsc{Find}\mathbf{N}(1^t, i)$. (See Parameter 5 in Definition 7.1.)
   (d) Find the constraint polynomial by computing $[P_t]^{\mathsf{A}} := \textsc{Find}\mathbf{P}(1^t)$.
   (e) Find the degree bounds for the constraint polynomial by computing $d_i^t := \textsc{Comp}\mathbf{D}(1^t, i)$ for $i = 0, 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$.
   (f) Find the basis for the linear subset S_t of $\mathbb{F}_t$ by computing $\big(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_t^{\mathbf{S}}(\mathsf{x})\big) := \textsc{Find}\mathbf{S}(1^t)$. (See Parameter 11 in Definition 7.1.) Set $m := \lceil \log |x| \rceil$ and define $S_m := \operatorname{span}\big(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_m^{\mathbf{S}}(\mathsf{x})\big)$.

2. *Generate a proof that A has low degree:*
   (a) Let p_0 be the evaluation table of A on $\mathbb{F}_t$:
   $$p_0 := \Big\{ \big(\alpha, A(\alpha)\big) \;:\; \alpha \in \mathbb{F}_t \Big\} \;.$$
   (b) Compute a proof of proximity π_0 for p_0 to $\mathrm{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathsf{m}_\mathbf{H}(t)} - 1)$:
   $$\pi_0 := P_{\mathrm{RS}}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, 2^{\mathsf{m}_\mathbf{H}(t)} - 1, A\big) \;.$$
3. *Generate a proof that A satisfies the constraint polynomial:*
   (a) Define the polynomial $B(x) := P_t\big(x, A(\mathsf{aff}_{t,1}(x)), \ldots, A(\mathsf{aff}_{t,\mathsf{c}_\mathbf{N}(t)}(x))\big)$.
   (b) Let p_1 be the evaluation table of B on $\mathbb{F}_t$:
   $$p_1 := \Big\{ \big(\alpha, B(\alpha)\big) \;:\; \alpha \in \mathbb{F}_t \Big\} \;.$$
   (c) Compute a proof of proximity π_1 for p_1 to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d_0^t + (2^{\mathsf{m}_\mathbf{H}(t)} - 1)\sum_{i=1}^{\mathsf{c}_\mathbf{N}(t)} d_i^t)$:
   $$\pi_1 := P_{\mathrm{VRS}}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{H_t}, d_0^t + (2^{\mathsf{m}_\mathbf{H}(t)} - 1)\textstyle\sum_{i=1}^{\mathsf{c}_\mathbf{N}(t)} d_i^t, B\big) \;.$$
4. *Generate a proof that A is consistent with $(x, 1^t)$:*
   (a) Define A_x to be the low-degree extension of the function $f\colon S_m \to \{0,1\}$ defined by $f(\alpha_i^{(x)}) := x_i$. (Padded with 0's if needed.)
   (b) Define $A' := A - A_x$.
   (c) Compute a proof of proximity π_c for the evaluation of A' to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, S_m, 2^{\mathsf{m}_\mathbf{H}(t)} - 1)$:
   $$\pi_\mathsf{c} := P_{\mathrm{VRS}}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, S_m, 2^{\mathsf{m}_\mathbf{H}(t)} - 1, A'\big) \;.$$
5. Set $\pi := (p_0, \pi_0, p_1, \pi_1, \pi_\mathsf{c})$ and output π.

The complexity analysis for P_{ACSP} is postponed to Lemma B.8.

The PCP prover P_{ACSP} from Construction 8.2 allows us to define the format of a PCP proof for membership into the language SuccinctACSP:

Definition 8.3 (PCP Proof for SuccinctACSP)**.** *A PCP proof of membership into* SuccinctACSP*, with respect to a choice*

$$\Big(f, (\mathsf{m}_\mathbf{H}, \mathsf{t}_\mathbf{H}, \mathsf{s}_\mathbf{H}, \mathbf{H}), (\mathsf{c}_\mathbf{N}, \mathsf{t}_\mathbf{N}, \mathsf{s}_\mathbf{N}, \mathbf{N}), (\mathsf{t}_\mathbf{D}, \mathsf{s}_\mathbf{D}, \mathbf{D}), (\mathsf{t}_\mathbf{P}, \mathsf{s}_\mathbf{P}, \mathsf{S}_\mathbf{P}, \mathsf{D}_\mathbf{P}, \mathbf{P}), (\mathsf{t}_\mathbf{S}, \mathsf{s}_\mathbf{S}, \mathbf{S})\Big)$$

of parameters, for an instance $(x, 1^t)$ is a string $\pi = (p_0, \pi_0, p_1, \pi_1, \pi_\mathsf{c})$, where:

1. *$p_0\colon \mathbb{F}_t \to \mathbb{F}_t$ is a function,*
2. *π_0 is a proof of proximity for p_0 to $\mathrm{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathsf{m}_\mathbf{H}(t)} - 1)$,*
3. *$p_1\colon \mathbb{F}_t \to \mathbb{F}_t$ is a function,*
4. *π_1 is a proof of proximity for p_1 to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d_0^t + (2^{\mathsf{m}_\mathbf{H}(t)} - 1)\sum_{i=1}^{\mathsf{c}_\mathbf{N}(t)} d_i^t)$, and*
5. *π_c is a proof of proximity for $p_0 - p_x$ to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, S_m, 2^{\mathsf{m}_\mathbf{H}(t)} - 1)$, for some function p_x and subspace S_m.*

Now we give the construction for the PCP verifier V_{ACSP} for SuccinctACSP. As already mentioned, the PCP verifier itself is an analogue of the verifier described in [BSS08, Proof of Theorem 3.2] (as well as the one left implicit in [BSGH+05, Theorem 2.5]), except that there are more details to ensure its efficiency and consistency with the instance. Its construction is as follows:

Construction 8.4 (PCP Verifier for SuccinctACSP)**.** The PCP verifier for SuccinctACSP, with respect to a choice

$$\Big(f, (\mathsf{m}_{\mathbf{H}}, \mathsf{t}_{\mathbf{H}}, \mathsf{s}_{\mathbf{H}}, \mathbf{H}), (\mathsf{c}_{\mathbf{N}}, \mathsf{t}_{\mathbf{N}}, \mathsf{s}_{\mathbf{N}}, \mathbf{N}), (\mathsf{t}_{\mathbf{D}}, \mathsf{s}_{\mathbf{D}}, \mathbf{D}), (\mathsf{t}_{\mathbf{P}}, \mathsf{s}_{\mathbf{P}}, \mathsf{S}_{\mathbf{P}}, \mathsf{D}_{\mathbf{P}}, \mathbf{P}), (\mathsf{t}_{\mathbf{S}}, \mathsf{s}_{\mathbf{S}}, \mathbf{S})\Big)$$

of parameters, is denoted V_{ACSP} and has hard-coded in it this choice of parameters. On input an instance $(x, 1^t)$ and with oracle access to a proof π following the format of Definition 8.3, the verifier V_{ACSP} does the following:

1. *Parameter instantiation:*
   (a) Generate the field extension $\mathbb{F}_t$ of $\mathbb{F}_2$: compute the degree $f(t)$, then compute the irreducible polynomial $I_t := \text{FindIrrPoly}\big(1^{f(t)}\big)$ with a root x, and let $\mathbb{F}_t := \mathbb{F}_2(\mathsf{x})$. (See Parameter 1 in Definition 7.1.) Note that a basis for $\mathbb{F}_t$ is given by $\mathcal{B}_{\mathbb{F}_t} := \big(1, \mathsf{x}, \ldots, \mathsf{x}^{f(t)-1}\big)$.
   (b) Find a basis for the linear subset H_t of $\mathbb{F}_t$ by computing $\mathcal{B}_{H_t} := \big(b_1^{(H_t)}, \ldots, b_{\mathsf{m}_{\mathbf{H}}(t)}^{(H_t)}\big) = \text{Find}\mathbf{H}(1^t)$. (See Parameter 3 in Definition 7.1.)
   (c) Find the affine functions $\{\mathsf{aff}_{t,i}(x) = a_{t,i} \cdot x + b_{t,i}\}_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)}$ that induce the "affine neighborhood" of each element in $\mathbb{F}_t$: for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, compute $(a_{t,i}, b_{t,i}) := \text{Find}\mathbf{N}(1^t, i)$. (See Parameter 5 in Definition 7.1.)
   (d) Find the degree bounds for the constraint polynomial by computing $d_i^t := \text{Comp}\mathbf{D}(1^t, i)$ for $i = 0, 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$.
   (e) Find the basis for the linear subset S_t of $\mathbb{F}_t$ by computing $\big(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_t^{\mathbf{S}}(\mathsf{x})\big) := \text{Find}\mathbf{S}(1^t)$. (See Parameter 11 in Definition 7.1.) Set $m := \lceil \log |x| \rceil$ and define $S_m := \text{span}\big(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_m^{\mathbf{S}}(\mathsf{x})\big)$.

2. *Proximity of p_0 to* RS*:*

   Invoke the (already "amplified") PCPP-verifier for the Reed-Solomon code from the second part of [BSS08, Theorem 3.2], with proximity parameter $\delta_{\text{RS}} = 1/(8\mathsf{c}_{\mathbf{N}}(t))$ and soundness $s'_{\text{RS}} := 1/2$, on explicit input $(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$, implicit input p_0, and proof of proximity π_0; that is, verify that

   $$V_{\text{aRS}}^{(p_0,\pi_0)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\text{RS}}, s'_{\text{RS}}\big)$$

   accepts.

3. *Proximity of p_1 to* VRS*:*

   Invoke the (already "amplified") PCPP-verifier for the vanishing Reed-Solomon code from the second part of [BSS08, Corollary 3/13], with proximity parameter $\delta_{\text{VRS}} := 1/8$ and soundness $s'_{\text{VRS}} := 1/2$, on explicit input $(\mathbb{F}_t, \mathbb{F}_t, H_t, d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t)$, implicit input p_1, and proof of proximity π_1; that is, verify that

   $$V_{\text{aVRS}}^{(p_1,\pi_1)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{H_t}, d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\textstyle\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\text{VRS}}, s'_{\text{VRS}}\big)$$

   accepts.

4. *Consistency between p_0 and p_1:*
   (a) Draw a random $\alpha \in \mathbb{F}_t$.
   (b) For $i = 1, \ldots, \mathsf{c_N}(t)$, compute $\alpha_i := \mathsf{aff}_{t,i}(\alpha) = a_{t,i} \cdot \alpha + b_{t,i}$.
   (c) Query p_0 at the following points: $\alpha_1, \ldots, \alpha_{\mathsf{c_N}(t)}$.
   (d) Query p_1 at the point α.
   (e) Compute $[P_t]^{\mathsf{A}} := \textsc{Find}\mathbf{P}(1^t)$.
   (f) Compute $\omega := [P_t]^{\mathsf{A}}\big(\alpha, p_0(\alpha_1), \ldots, p_0(\alpha_{\mathsf{c_N}(t)})\big)$. (See Parameter 9 in Definition 7.1.)
   (g) Verify that $p_1(\alpha) = \omega$.

5. *Consistency of p_0 with the instance $(x, 1^t)$:*

   Define A_x to be the low-degree extension of the function $f\colon S_m \to \{0,1\}$ defined by $f(\alpha_i^{(x)}) := x_i$; let p_x be the evaluation of A_x on $\mathbb{F}_t$.

   Invoke the (already "amplified") PCPP-verifier for the vanishing Reed-Solomon code from the second part of [BSS08, Corollary 3/13], with proximity parameter $\delta_{\mathsf{c}} := 1/(8\mathsf{c_N}(t))$ and soundness $s'_{\mathsf{c}} := 1/2$, on explicit input $(\mathbb{F}_t, \mathbb{F}_t, S_m, 2^{\mathsf{m_H}(t)} - 1)$, implicit input $p_0 - p_x$, and proof of proximity π_{c}; that is, verify that

$$V_{\mathrm{aVRS}}^{(p_0 - p_x, \pi_{\mathsf{c}})}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, S_m, 2^{\mathsf{m_H}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}}\big)$$

   accepts.

The complexity analysis for V_{ACSP} is postponed to Lemma B.8.

Now we prove the correctness of the constructions:

Proof of Theorem 8.1. We prove here that the PCP verifier V_{ACSP} from Construction 8.4 has perfect completeness and soundness 1/2. The analysis of the query complexity, randomness complexity, proof length complexity, prover running time, and verifier running time is carried out in detail in Section B (see especially Lemma B.8).

Completeness. First, we begin by proving that V_{ACSP} has perfect completeness. So suppose that $(x, 1^t) \in \textsc{SuccinctACSP}$, and let $A\colon \mathbb{F}_t \to \mathbb{F}_t$ be an assignment polynomial that witnesses this.

Let $p_0\colon \mathbb{F}_t \to \mathbb{F}_t$ be the evaluation on $\mathbb{F}_t$ of the assignment polynomial A. Recall that, by Definition 7.1, the degree of A is required to be less than $2^{\mathsf{m_H}(t)}$; hence, $p_0 \in \mathrm{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathsf{m_H}(t)} - 1)$. Invoking the completeness property of the PCPP verifier V_{aRS}, we deduce that there exists a string π_0 that makes $V_{\mathrm{aRS}}^{(p_0,\pi_0)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 2^{\mathsf{m_H}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}}\big)$ accept with probability 1. We conclude that the same string π_0 makes the first subtest of V_{ACSP} accept with probability 1, because the first subtest of V_{ACSP} is simply the test $V_{\mathrm{aRS}}^{(p_0,\pi_0)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 2^{\mathsf{m_H}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}}\big)$. (See Step 2 of V_{ACSP}.)

Let $p_1\colon \mathbb{F}_t \to \mathbb{F}_t$ be the evaluation on $\mathbb{F}_t$ of the polynomial $B\colon \mathbb{F}_t \to \mathbb{F}_t$ defined as follows:

$$B(x) := P_t\Big(x, A\big(\mathsf{aff}_{t,1}(x)\big), \ldots, A\big(\mathsf{aff}_{t,\mathsf{c_N}(t)}(x)\big)\Big) \ .$$

Notice that the degree of B can be upper bounded as follows:

$$\deg(B) \le \deg_{x_0}(P_t) + \deg(A) \cdot \sum_{i=1}^{\mathsf{c_N}(t)} \deg_{x_i}(P_t) \le d_0^t + \big(2^{\mathsf{m_H}(t)} - 1\big) \sum_{i=1}^{\mathsf{c_N}(t)} d_i^t \ ;$$

furthermore, because A is a witness for $(x, 1^t) \in \textsc{SuccinctACSP}$, due to Item (i) in Definition 7.1, we know that B vanishes on the linear subset H_t; hence, $p_1 \in \mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d_0^t + (2^{\mathsf{m_H}(t)} - 1)\sum_{i=1}^{\mathsf{c_N}(t)} d_i^t)$. Invoking the completeness property of the PCPP verifier V_{aVRS}, we deduce that there exists a string π_1 that makes $V_{\mathrm{aVRS}}^{(p_1,\pi_1)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{H_t}, d_0^t + (2^{\mathsf{m_H}(t)} - 1)\sum_{i=1}^{\mathsf{c_N}(t)} d_i^t, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}}\big)$ accept with probability 1. We conclude that the same string π_1 makes the second subtest of V_{ACSP} accept with probability 1, because the second subtest of V_{ACSP} is simply the test $V_{\mathrm{aVRS}}^{(p_1,\pi_1)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{H_t}, d_0^t + (2^{\mathsf{m_H}(t)} - 1)\sum_{i}^{\mathsf{c}}$ (See Step 3 of V_{ACSP}.)

Moreover, by construction of p_0 and p_1, for every $\alpha(\mathsf{x}) \in \mathbb{F}_t$,

$$\begin{aligned} p_1\big(\alpha(\mathsf{x})\big) =& B\big(\alpha(\mathsf{x})\big) \\ =& P_t\Big(\alpha(\mathsf{x}), A\big(\mathsf{aff}_{t,1}(\alpha(\mathsf{x}))\big), \ldots, A\big(\mathsf{aff}_{t,\mathsf{c_N}(t)}(\alpha(\mathsf{x}))\big)\Big) \\ =& P_t\Big(\alpha(\mathsf{x}), p_0\big(\mathsf{aff}_{t,1}(\alpha(\mathsf{x}))\big), \ldots, p_0\big(\mathsf{aff}_{t,\mathsf{c_N}(t)}(\alpha(\mathsf{x}))\big)\Big) \end{aligned}$$

so that the third subtest of V_{ACSP} succeeds with probability 1. (See Step 4 of V_{ACSP}.)

Finally, because A is a witness for $(x, 1^t) \in \textsc{SuccinctACSP}$, due to Item (ii) in Definition 7.1, we know that A is consistent with the instance $(x, 1^t)$, i.e., we know that, for $i = 1, \ldots, |x|$, $A(\alpha_i^{(x)}(\mathsf{x})) = x_i$ where $\alpha_i^{(x)}(\mathsf{x})$ is the i-th element of $S_m := \mathrm{span}\big(\alpha_1^{\mathsf{s}}(\mathsf{x}), \ldots, \alpha_m^{\mathsf{s}}(\mathsf{x})\big)$, where $\big(\alpha_1^{\mathsf{s}}(\mathsf{x}), \ldots, \alpha_t^{\mathsf{s}}(\mathsf{x})\big)$ is a basis for S_t. Hence, letting A_x be the low-degree extension of the function $f\colon S_m \to \{0,1\}$ defined by $f(\alpha_i^{(x)}) := x_i$, and p_x the evaluation of A_x on $\mathbb{F}_t$, we know that $p_0 - p_x$ is in $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, S_m, 2^{\mathsf{m_H}(t)} - 1)$. Once again invoking the completeness property of the PCPP verifier V_{aVRS}, we deduce that there exists a string π_{c} that makes $V_{\mathrm{aVRS}}^{(p_0-p_x,\pi_{\mathsf{c}})}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, S_m, 2^{\mathsf{m_H}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}}\big)$ accept with probability 1. We conclude that the same string π_{c} makes the fourth subtest of V_{ACSP} accept with probability 1, because the fourth (and last) subtest of V_{ACSP} is simply the test $V_{\mathrm{aVRS}}^{(p_0-p_x,\pi_{\mathsf{c}})}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, S_m, 2^{\mathsf{m_H}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}}\big)$. (See Step 5 of V_{ACSP}.)

Thus, we have established that V_{ACSP} has perfect completeness. (And note that the PCP prover P_{ACSP} from Construction 8.2, on input $(x, 1^t)$ and A, does indeed construct the proof $\pi = (p_0, \pi_1, p_1, \pi_1, \pi_{\mathsf{c}})$ that we used to make V_{ACSP} accept with probability 1 — by providing the correct evaluation tables p_0 and p_1 and running the PCPP provers for RS and VRS with the appropriate inputs.)

Soundness. Next, we prove that V_{ACSP} has soundness $1/2$. As before, let $S_m := \mathrm{span}\big(\alpha_1^{\mathsf{s}}(\mathsf{x}), \ldots, \alpha_m^{\mathsf{s}}(\mathsf{x})\big)$, A_x be the low-degree extension of the function $f\colon S_m \to \{0,1\}$ defined by $f(\alpha_i^{(x)}) := x_i$, and p_x the evaluation of A_x on $\mathbb{F}_t$.

Suppose that $(x, 1^t) \notin \textsc{SuccinctACSP}$. We distinguish between three cases:

- *Case 1:* the function p_0 is $1/(8\mathsf{c_N}(t))$-far from $\mathrm{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathsf{m_H}(t)} - 1)$.

  Recall that the first subtest of V_{ACSP} (Step 2) tests proximity of p_0 to $\mathrm{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathsf{m_H}(t)} - 1)$ using the (amplified) RS verifier V_{aRS} with proximity parameter $\delta_{\mathrm{RS}} = (8\mathsf{c_N}(t))^{-1}$ and soundness $s'_{\mathrm{RS}} = 1/2$; thus, in this case the first subtest of V_{ACSP} rejects with probability at least $1/2$.

- *Case 2:* the function p_1 is 1/8-far from $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d_0^t + (2^{\mathsf{m_H}(t)} - 1)\sum_{i=1}^{\mathsf{c_N}(t)} d_i^t)$.

  Recall that the second subtest of V_{ACSP} (Step 3) tests proximity of p_1 to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d_0^t + (2^{\mathsf{m_H}(t)} - 1)\sum_{i=1}^{\mathsf{c_N}(t)} d_i^t)$ using the (amplified) VRS verifier V_{aVRS} with proximity

parameter $\delta_{\text{VRS}} = 1/8$ and soundness $s'_{\text{VRS}} = 1/2$; thus, in this case the second subtest of V_{ACSP} rejects with probability at least $1/2$.

- *Case 3:* the function $p_0 - p_x$ is $1/(8\mathsf{c}_{\mathbf{N}}(t))$-far from $\text{VRS}(\mathbb{F}_t, \mathbb{F}_t, S_m, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$.

  Recall that the fourth subtest of V_{ACSP} (Step 5) tests proximity of $p_0 - p_x$ to $\text{VRS}(\mathbb{F}_t, \mathbb{F}_t, S_m, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$ using the (amplified) VRS verifier V_{aVRS} with proximity parameter $\delta_{\mathsf{c}} = 1/(8\mathsf{c}_{\mathbf{N}}(t))$ and soundness $s'_{\mathsf{c}} = 1/2$; thus, in this case the second subtest of V_{ACSP} rejects with probability at least $1/2$.

- *Case 4:* neither of the above two cases hold.

  Let $A\colon \mathbb{F}_t \to \mathbb{F}_t$ be a polynomial of degree less than $2^{\mathsf{m}_{\mathbf{H}}(t)}$ whose evaluation table over $\mathbb{F}_t$ is closest to p_0, and let $B\colon \mathbb{F}_t \to \mathbb{F}_t$ be the polynomial defined as $B(x) := P_t\big(x, A(\mathsf{aff}_{t,1}(x)), \ldots, A(\mathsf{aff}_{t,\mathsf{c}_{\mathbf{N}}(t)}(x))\big)$. Note that A is unique, because $1-(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)/|\mathbb{F}_t|$ is larger than $2\delta_{\text{RS}} = 1/(4\mathsf{c}_{\mathbf{N}}(t))$. (See Lemma 3.11.)

  Observe that $p_0 - p_x$ is $1/(8\mathsf{c}_{\mathbf{N}}(t))$-close to the evaluation table of $A - A_x$ over $\mathbb{F}_t$. Let $A'\colon \mathbb{F}_t \to \mathbb{F}_t$ be a polynomial of degree less than $2^{\mathsf{m}_{\mathbf{H}}(t)}$ vanishing on S_m whose evaluation table over $\mathbb{F}_t$ is closest to $p_0 - p_x$. Note that $A' = A - A_x$ because both A' and $A - A_x$ are polynomials of degree less than $2^{\mathsf{m}_{\mathbf{H}}(t)}$ that are $1/(8\mathsf{c}_{\mathbf{N}}(t))$-close to $p_0 - p_x$ and $1 - (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)/|\mathbb{F}_t|$ is larger than $1/(4\mathsf{c}_{\mathbf{N}}(t))$. (See Lemma 3.11.) Thus we deduce that A is consistent with the instance $(x, 1^t)$, i.e., satisfies Item (ii) in Definition 7.1.

  Define the function $p_2\colon \mathbb{F}_t \to \mathbb{F}_t$ by

$$p_2(x) := P_t\Big(x, p_0\big(\mathsf{aff}_{t,1}(x)\big), \ldots, p_0\big(\mathsf{aff}_{t,\mathsf{c}_{\mathbf{N}}(t)}(x)\big)\Big) \ .$$

  Observe that, for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$ and a random $\alpha(\mathsf{x})$ in $\mathbb{F}_t$, since $\mathsf{aff}_{t,i}$ is an affine function, $\mathsf{aff}_{t,i}(\alpha(\mathsf{x}))$ is a random element in $\mathbb{F}_t$. We deduce then, via a union bound (on the fact that p_0 is $1/(8\mathsf{c}_{\mathbf{N}}(t))$-close to the evaluation table of A over $\mathbb{F}_t$), that p_2 must be $1/8$-close to the evaluation table of $B(x)$ over $\mathbb{F}_t$.

  Now let $B'\colon \mathbb{F}_t \to \mathbb{F}_t$ be a polynomial of degree at most $d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t$ vanishing on H_t whose evaluation table over $\mathbb{F}_t$ is closest to p_1. Again note that B' is unique, because $1 - (d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t)/|\mathbb{F}_t|$ is larger than $2\delta_{\text{VRS}} = 1/4$. (See Lemma 3.11.) Now, B and B' must be distinct for, otherwise, B would vanish on H_t, implying that A would satisfy the constraint polynomial P_t, which would in turn imply (together with the fact that A is consistent with the instance $(x, 1^t)$) that $(x, 1^t) \in \text{SuccinctACSP}$ — a contradiction. Thus, since both B and B' are (distinct) polynomials of degree at most $d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t \le |\mathbb{F}_t|/4$, their evaluation tables over $\mathbb{F}_t$ may agree on at most a $1/4$ fraction of their entries. Thus, by a union bound, the third subtest of V_{ACSP} (Step 4) accepts with probability at most $1/8 + 1/8 + 1/4 = 1/2$.

This completes the proof of the soundness property. □

Remark 8.5. More generally, the soundness analysis would have also worked for any proximity parameter δ_{RS} for V_{aRS} and proximity parameter δ_{VRS} for V_{aVRS} as long as the following conditions hold:

$$1 - \frac{2^{\mathsf{m}_{\mathbf{H}}(t)} - 1}{|\mathbb{F}_t|} > 2\delta_{\text{RS}} \ , \qquad \text{(to ensure uniqueness of } A\text{)}$$

$$1 - \frac{2^{\mathsf{m}_{\mathbf{H}}(t)} - 1}{|\mathbb{F}_t|} > 2\delta_{\mathsf{c}} \ , \qquad \text{(to ensure uniqueness of } A' \text{ and } A' = A - A_x\text{)}$$

$$1 - \frac{d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t}{|\mathbb{F}_t|} > 2\delta_{\mathrm{VRS}} \ , \qquad \text{(to ensure uniqueness of } B'\text{)}$$

$$\mathsf{c}_{\mathbf{N}}(t)\delta_{\mathrm{RS}} + \delta_{\mathrm{VRS}} + \frac{d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t}{|\mathbb{F}_t|} \leq \frac{1}{2} \ .$$

For example, for the specific choices of $\delta_{\mathrm{RS}} = \delta_{\mathsf{c}} = (8\mathsf{c}_{\mathbf{N}}(t))^{-1}$ and $\delta_{\mathrm{VRS}} = 1/8$ that we used in the proof of Theorem 8.1, all of the above constraints are implied by the SuccinctACSP constraint from Parameter 7 of Definition 7.1.

Remark 8.6. Using *affine functions* to select neighbors of a field element is important in the proof of Theorem 8.1 in two ways.

From the perspective of completeness (and efficiency), if one were to use functions that when represented as polynomials have degree more than 1, the polynomial B, resulting from the "composition" of the assignment polynomial A with the constraint polynomial P_t and the neighbor functions, would have degree at least quadratic in the degree of A. Thus completeness would hold only if we would enlarge the field $\mathbb{F}_t$ proportionally, but this would induce proof sizes of quadratic length, which is too much.

From the perspective of the soundness proof, it is important that the neighbor functions preserve the uniform distribution, so that we are able to deduce (via the union bound) a bound on the distance of p_2 to the evaluation table of B from only a bound on the distance of p_0 to A.

Using neighbor functions that are affine functions simultaneously satisfies both of the above needs.

Remark 8.7. The consistency with the instance requirement in the definition of SuccinctACSP (i.e., Item (ii) of Definition 7.1) is quite "specialized": it specifically requires that the concatenation of the $A(\alpha_i(\mathsf{x}))$, for $i = 1, \ldots, |x|$, is, when properly converted to bits, equal to x.

This very simple form of consistency check enables us to use a very simple test for testing this requirement: namely, we can use a simple test of proximity to an appropriate Vanishing Reed-Solomon code. (See Step 5 of V_{ACSP}.)

However, being able to support more general consistency tests would generalize the definition of SuccinctACSP, and thereby slightly simplify reducing to it. For example, one could consider giving both x and $(A(\alpha_1(\mathsf{x})), \ldots, A(\alpha_{|x|}(\mathsf{x})))$ to a test procedure that decides whether the two are "consistent" or not (and presumably such a test would have the freedom to "extract" arbitrary information from each $A(\alpha_i(\mathsf{x}))$ and then perform some test).

But, to support these more general tests, we would need to specify the complexity of the test when understood as a polynomial, and perform more complicated tests similar to the one we performed for checking the constraint polynomial P_t. We do not see any reason that calls for such greater generality in light of the additional complexity that would arise from supporting these more general tests.

Remark 8.8. While the techniques used to construct the PCP for SuccinctACSP can be used to also construct a PCP of Proximity [DR04, BSGH+06] for SuccinctACSP, we do not work out the details.

Indeed, suppose that the instance $(x, 1^t)$ is such that x is long, and one wishes to separate an offline and online phases of a verifier, where during the offline phase the verifier may run in

time proportional to $|x|$, but not during the online phase. In such a case PCPPs could help, *but* there is a better and simpler solution: one can use the *Untrusted Input Lemma* reduction to a random-access machine before reducing to SuccinctACSP, as discussed in [BSCGT12].

Indeed, PCPs of Proximity have the disadvantage that one must have oracle access to the input (which is often inconvenient), whereas in the method we suggest all is required is to remember a short hash of the input. Moreover, from an efficiency standpoint, enhancing our construction to a PCP of Proximity would make it somewhat less efficient.

So from both a convenience and efficiency perspectively, the reduction of [BSCGT12] is preferred. (Though it introduces computational assumptions, and thus only obtain a probabilistically-checkable *argument*.)

Remark 8.9. Ben-Sasson and Sudan constructed PCPs for an algebraic constraint satisfaction problem [BSS08, Section 3.3.1], which they call ALGEBRAIC-CSP$_{k,d}$, as part of the proof of their main theorem [BSS08, Theorem 2.2], which gives PCPs for $\mathsf{NTIME}(t)$ languages with quasilinear length complexity and polylogarithmic randomness and query complexity. Our construction follows their approach, but necessitates additional details to work for the more general language SuccinctACSP and to ensure that the verifier is efficient and, moreover, only runs quasilinearly relative to the instance size. In fact, the construction of this section is a generalization of a theorem left implicit in [BSGH+05], where a special case of the language SuccinctACSP was considered for the purpose of constructing efficient verifiers for the PCPs of Ben-Sasson and Sudan. In our case, we also need to concentrate on the construction of the prover, to ensure that it runs in quasilinear time, and the complexity of the verifier in terms of the input instance (which we will also ensure is quasilinear); these two concerns were not addressed before, and are ultimately crucial to prove our Theorem 1.

Remark 8.10 (combinatorial techniques)**.** While the early results about PCPs exploited predominantly algebraic techniques [BFLS91, AS98, ALM+98], by now also a combinatorial proof is known [Din07]. However, despite the recent improvements in the understanding of verifier-efficient PCPs [Mei09], combinatorial techniques seem to still significantly lag behind algebraic techniques in potential for practical PCPs. For example, it is still not known how to generate quasilinear-size PCP proofs using combinatorial techniques (though recent progress was shown by [Mei12]); moreover, the current understanding of practical implementations of expander constructions and algorithmic implementations of graph operations does not stand up to the current understanding of efficient algebraic computation. Indeed, when it comes to polynomial arithmetic, we already possess a quite good understanding of how to multiply, evaluate, interpolate very fast in practice via FFT methods. This algorithmic knowledge is also crucial for practical implementations of PCPs, and indeed we make explicit use of it in the algorithms we present.

9 Proof of Theorem 1

In this section we prove Theorem 1, which was discussed and formally stated in Section 2.2.

High-level strategy. It is useful to break the problem of constructing PCPs for SCS with stringent efficiency constraints into *two* different subproblems: (a) constructing a PCP system for some "PCP-friendly" NEXP-complete problem $\mathcal{P}$ with a small concrete-efficiency threshold (relative to $\mathcal{P}$), and (b) constructing tight reductions from, in our case, SCS to $\mathcal{P}$. Indeed, PCPs are only known to exist for certain problems having strong algebraic or combinatorial "structure"; such problems are quite "unnatural" and, without tight reductions from SCS to $\mathcal{P}$, we would have $B = \infty$.[11]

Typically, one also defines a flexible "interface" between the two aforementioned subproblems (a) and (b), in order to create some "modularity". Following and generalizing [BSGH+05], we thus introduce a family of *succinct* algebraic constraint satisfaction problems, which we call SuccinctACSP (see Section 7 for details), attempting to *simultaneously* capture the essential ingredients that make a problem amenable to ("direct") probabilistic checking as well as be general enough to make it easy to reduce from less structured (natural) problems (such as SCS).[12] In other words, we choose $\mathcal{P} :=$ SuccinctACSP. This choice "decouples" (a) and (b), and progress can be made independently on each subproblem.[13] In light of the aforementioned decoupling, our proof of Theorem 1 relies on two ingredients: (a) an algebraic PCP construction for SuccinctACSP (see Section 8 for details), and (b) the use of reductions of Ben-Sasson et al. [BSCGT12] from random-access machines to SuccinctACSP to obtain a reduction from SCS to SuccinctACSP. We will ensure that both of these ingredients meet the required efficiency constraints for obtaining a PCP system for SCS with $B < \infty$.

Proof of Theorem 1. It suffices to construct a PCP system for SCS where the prover runs in time $|F|2^n\mathsf{poly}(n)$ and the verifier runs in time $|F|\mathsf{poly}(n)$.

Step 1. We first invoke the reductions of Ben-Sasson et al. [BSCGT12]. Concretely, Ben-Sasson et al. study reductions from *bounded-halting problems on random-access machines*, denoted bhRAM, to SuccinctACSP; specifically, for a given random-access machine M, bhRAM(M) is the language of pairs (x, T) such that the random-access machine M non-deterministically accepts x within T steps. A reduction from bhRAM to SuccinctACSP is an efficient transformation from the given machine M to par^M such that $(x, T) \in$ bhRAM(M) if and only if $(x, T) \in$ SuccinctACSP(par^M), where $\mathsf{par}^M = \{\mathsf{par}^M_T\}_{T\in\mathbb{N}}$ is a family of parameter choices for SuccinctACSP; that is, par^M_T describes the algebraic constraints to be used for verifying membership of the instance (x, T). In this case, the cost of the reduction is naturally measured as the size growth rate of the size of the field $\mathbb{F}_T$.

Crucial to us is that:

- The efficiency of their reduction is *quasilinear*, in the sense that $|\mathbb{F}_T| = T \cdot S_M \cdot \mathsf{polylog}(T)$, where S_M is the "processor complexity" of M as a Boolean circuit. We can ensure to work only with "reasonable" machines M, for which $S_M = \mathsf{polylog}(T)$.

[12] We focus on *algebraic PCPs*. Present PCP technology suggests greater flexibility and understanding of computational aspects of such PCPs; e.g., we know of quasilinear-size PCPs only via algebraic techniques [BSS08], and have a good picture of computational aspects of algebra [vzGG03]. (See Remark 8.10.)

[13] We think that such a decoupling is a significant simplification of the problem of constructing practical PCPs in general, given the amount of work that goes towards satisfying solutions of either subproblem and given that the two subproblems are different in flavor and thus demand for quite different sets of techniques.

- The instance size does not increase too much; in fact, the instance is not changed at all throughout the reduction.

We can then deduce from the above the existence of a quasilinear reduction from SCS to SuccinctACSP that preserves the instance: we can simply fix the random-access machine M^* that, on input a circuit description F, will non-deterministically verify that $F \in \text{SCS}$; note that this can be done in time $O(|F| \cdot 2^n)$ when F describes a circuit of size 2^n.[14]

Step 2. Having applied the reductions of [BSCGT12] to the machine M^*, we obtain a choice of parameters par^{M^*} for SuccinctACSP, and we can now simply invoke Theorem 8.1, which is our PCP construction for SuccinctACSP with the requisite efficiency properties. Specifically, in our PCP construction:

- the prover runs in time that is quasilinear in the instance and in the field size;
- the verifier runs in time that is quasilinear in the instance and polylogarithmic in the field size.

Putting the two steps together, we obtain a PCP system for SCS where the prover runs in time $|F|2^n\mathsf{poly}(n)$ and the verifier runs in time $|F|\mathsf{poly}(n)$, as desired. □

[14] We take the opportunity to mention that a "direct" reduction from SCS to SuccinctACSP through a naive use of the techniques contained in [BSCGT12] will not work. Indeed, if we were to simply route the circuit represented by F using De Bruijn graph (in a similar manner as in [PS94], by also paying attention to the succinctness of the reduction) and then arithmetize the resulting coloring problem, it will not work; the reason is that the degree of F when viewed as a polynomial may be too high and cause the reduction to be too expensive (e.g., quadratic). The "right" way to do this without relying on random-access machines would be to route a universal circuit that performs a computation similar to th random-access machine we chose; by ensuring that the universal circuit has a highly-structured topology, the description of the universal circuit would be shallow, and thus this description, when viewed as a polynomial, would have low enough degree for the reduction to work out. But this direct reduction would be unnecessarily complicated, and taking a path, as we do, through random-access machines is more natural and cleaner.

10 Proof of Theorem 3

Proof of Theorem 3. Because the reductions of [BSCGT12] preserve the properties we are going to prove (see Remark 10.1), it suffices to concentrate on our PCP construction for SuccinctACSP (see Theorem 8.1).

Also, it suffices to prove that our PCP construction for SuccinctACSP can be used to construct almost universal arguments, because [Mic00, Val08, CT10, BCCT11] only need a subset of the properties needed for almost universal arguments. We thus proceed as follows.

We formally introduce universal arguments, along with our relaxation of *almost* universal arguments (Section 10.1). We then prove that the PCPs from Theorem 8.1 have the relatively-efficient oracle construction property (Section 10.2), have the non-adaptive verifier property (Section 10.3), have the efficient reverse-sampling property (Section 10.4), and the "explicit" proof of knowledge property (Section 10.5). Finally, we show how to construct almost universal arguments using PCPs with these four properties (Section 10.6). □

Remark 10.1 (choice of computation model)**.** It suffices to prove the four required properties for our PCP system for (an arbitrary choice of parameters for) SuccinctACSP, because any reasonable Levin reduction to SuccinctACSP will preserve the four properties;[15] in particular, so will the Levin reductions from bounded halting problem on RAMs studied by Ben-Sasson et al. [BSCGT12]. Therefore, our focus will be on the PCP system $(P_{\text{ACSP}}, V_{\text{ACSP}})$ that we construct for SuccinctACSPs, and so we will prove our results relative to it, assuming that any reductions have already been applied by both the prover and verifier. We thus fix an arbitrary choice of parameters for SuccinctACSP.

Remark 10.2 (examples of positive applications)**.** A succinct argument is a computationally-sound interactive proof enabling a verifier to check membership of an instance y in an NP language L in time that is bounded by $p(\kappa, |y|, \log t)$, where t is the time to (non-deterministically) evaluate the NP verification relation for L on input y, p is a fixed polynomial independent of L, and κ is a security parameter that determines the soundness error. Universal arguments are succinct arguments where soundness is strengthened to a certain proof-of-knowledge property.

Succinct arguments have been used to achieve cryptographic tasks that could be very useful in practice. Most such "positive" applications of succinct arguments only consider statements that lie in NP; hence, such applications are fortunately *not* affected by the technical difference between universal arguments and almost universal arguments. (Recall that with our PCPs we are only able to construct almost universal arguments.)

Theorem 3 can thus be interpreted as alleviating the "succinct-argument bottleneck" that is currently preventing real-world practicality of many desirable cryptographic constructions. Examples of such constructions include:

- **Delegation of computation.** In a *delegation of computation scheme*, a weak party wishes to delegate the computation of a function F on an input x to a another (untrusted) more powerful party (who may or may not be allowed to contribute his own input to the computation). If one insists on avoiding expensive offline precomputations, succinct arguments are essentially the only tool for constructing such schemes.[16]

[15] In fact even *computational* Levin reductions, when properly defined, will preserve these properties; see [BSCGT12].

[16] Sometimes one is also interested in a privacy property as well. Such a property can be achieved by appropriately combining the succinct argument with a fully-homomorphic encryption scheme [Gen09]. Fully-homomorphic

- **Proof-carrying data.** Chiesa and Tromer [CT10] have shown how to construct non-interactive succinct arguments of knowledge that can be "recursively composed",[17] using any (constant-round public-coins) succinct argument in a model where parties have access to a signature functionality. They then show how to use these to construct a *proof-carrying data system*, a cryptographic primitive that is able to dynamically compile a distributed computation into one where a given "local" security property is enforced. Proof-carrying data systems naturally address a number of integrity concerns that arise in systems security, and it would be great if such a powerful primitive could be made more practical.
- **Computationally-sound checkers.** Micali [Mic98] showed how non-interactive succinct arguments that are publicly verifiable give rise to *computationally-sound checkers*, a cryptographic primitive that is capable of certifying NP heuristics.

Succinct arguments are also one of the most powerful and beautiful combinations of complexity-theoretic tools (such as PCPs) and cryptographic tools (such as collision-resistant hash functions), and investigating their efficiency is a very exciting research direction with great impact potential to practice.

Remark 10.3 (computational overheads)**.** The construction of universal arguments starting from a PCP system (with certain properties) and a collision-resistant hash-function family [BG02] introduces very little additional computational overhead on top of the PCP system.

Therefore, establishing that almost universal arguments can be constructed (via the same construction as in [BG02]) using a PCP system with a certain concrete-efficiency threshold B essentially means constructing almost universal arguments with a concrete-efficiency threshold B' not much greater than B.[18]

Other constructions of succinct arguments starting from PCPs are not as "light"; for example, in [BCCT11, DFH11, GLR11], one must also use a private information retrieval scheme.

10.1 Universal Arguments and Our Relaxation

Barak and Goldreich [BG02] relax the computational soundness condition of the (interactive) CS proofs of Micali [Mic00], and show how to construct *universal arguments* under the assumption that *standard* collision-resistant hashing schemes exist (improving over the results of Kilian [Kil92] and Micali [Mic00], where *strong* collision-resistant hashing schemes were needed instead).

Specifically, [BG02] consider a language analogous to the CS language of Micali, defined as follows (once adapted to the case of random-access machines):

Definition 10.4 (Universal Set)**.** *The* **universal set***, denoted $S_{\mathcal{U}}$, is the set of all triples $y = (M, x, T)$ such that M is (the description of) a two-tape random-access machine, x is a binary string, and T is a binary integer such that there is a binary string w for which M accepts (x, w) within T steps (where x is written on the input tape and w on the witness tape). We denote by $R_{\mathcal{U}}$ the witness relation of the universal set $S_{\mathcal{U}}$, and by $R_{\mathcal{U}}(y)$ the set of valid witnesses for a given triple y.*

encryption is another computationally-heavy primitive, whose efficiency has been studied by, e.g., [GH11, Gen11]. However, quasilinear reductions that are compatible with fully-homomorphic encryption are not known; see [BSCGT12].

[17] Also called *succinct hearsay arguments*: succinct arguments that can prove statements based on "hearsay", namely, based on previous proofs.

[18] It is of course possible to extend the discussion of Section 2.1 about concrete-efficiency thresholds to the case of succinct arguments.

The name "universal" comes from the fact that every language L in NP is linear-time reducible to $S_{\mathcal{U}}$ by mapping every instance x to the triple $(M_L, x, 2^{|x|})$, where M_L a two-tape random-access machine that decides L, so $S_{\mathcal{U}}$ can uniformly handle all NP statements.

A *universal argument* is simply an efficient interactive argument of knowledge for the universal set:

Definition 10.5. *A* **universal argument system** *is a pair of machines $(P_{\mathsf{UA}}, V_{\mathsf{UA}})$ that satisfies the following conditions:*

1. Efficient verification: *There exists a polynomial p such that for any $y = (M, x, T)$, the total time spent by the (probabilistic) verifier strategy V_{UA}, on common input y, is at most $p(|y|) = p(|M| + |x| + \log T)$. In particular, all messages exchanged during the interaction have length that is at most $p(|y|)$.*

2. Completeness via a relatively-efficient prover: *For every $\big((M, x, T), w\big) \in R_{\mathcal{U}}$,*

$$\Pr\Big[\langle P_{\mathsf{UA}}(w), V_{\mathsf{UA}}\rangle(M, x, T) = 1\Big] = 1 \ .$$

   Furthermore, there exists a polynomial p such that for every $((M, x, T), w) \in R_{\mathcal{U}}$ the total time spent by $P_{\mathsf{UA}}(w)$, on common input (M, x, T), is at most

$$p\Big(|M| + |x| + \mathsf{time}_M(x, w)\Big) \leq p\big(|M| + |x| + T\big) \ .$$

3. Computational soundness: *For every family of polynomial-size prover circuits $\{\widetilde{P}_\kappa\}_{\kappa\in\mathbb{N}}$ and every positive constant c, for all sufficiently large $\kappa \in \mathbb{N}$, for every $(M, x, T) \in \{0,1\}^\kappa - S_{\mathcal{U}}$,*

$$\Pr\Big[\langle \widetilde{P}_\kappa, V_{\mathsf{UA}}\rangle(M, x, T) = 1\Big] < \frac{1}{\kappa^c} \ .$$

4. Weak proof of knowledge: *Implies computational soundness, and is stated below in Definition 10.6.*

Definition 10.6 (Weak Proof of Knowledge)**.** *For every two positive polynomials s_{UA} and p_{UA} there exist a positive polynomial q_{UA} and a probabilistic polynomial-time weak knowledge extractor E_{UA} such that for every family of s_{UA}-size prover circuits $\widetilde{P}_{\mathsf{UA}} = \{\widetilde{P}_{\mathsf{UA},\kappa}\}_{\kappa\in\mathbb{N}}$, for all sufficiently large $\kappa \in \mathbb{N}$, for every instance $y = (M, x, T) \in \{0,1\}^\kappa$ the following holds:*

Suppose that the prover circuit $\widetilde{P}_{\mathsf{UA},\kappa}$ convinces V_{UA} to accept y with probability greater than $p_{\mathsf{UA}}(\kappa)^{-1}$ (taken over a random choice of internal randomness for V_{UA}).

Then, with probability greater than $q_{\mathsf{UA}}(\kappa)^{-1}$ taken over a random choice of internal randomness r for E_{UA}, the weak knowledge extractor E_{UA}, with oracle access to the code of $\widetilde{P}_{\mathsf{UA},\kappa}$ and on input y, is an implicit representation of a valid witness w for y.

In symbols:

$$\forall\, s_{\mathsf{UA}}\ \forall\, p_{\mathsf{UA}}\ \exists\, q_{\mathsf{UA}}\ \exists\, E_{\mathsf{UA}}\ \forall\, s_{\mathsf{UA}}\text{-}size\ \widetilde{P}_{\mathsf{UA}}\ \exists\, K\ \forall\, \kappa > K\ \forall\, y = (M, x, T) \in \{0,1\}^\kappa$$

if

$$\Pr\Big[\langle \widetilde{P}_{\mathsf{UA},\kappa}, V_{\mathsf{UA}}\rangle(y) = 1\Big] > \frac{1}{p_{\mathsf{UA}}(\kappa)} \ ,$$

then

$$\Pr_r\Big[\ \textit{there exists } w = w_1 \cdots w_T \in R_{\mathcal{U}}(y) \textit{ such that, } \forall\, i \in [T]\,,\ E_{\mathsf{UA}}^{\langle \widetilde{P}_{\mathsf{UA},\kappa}\rangle}(y, i\,; r) = w_i\Big] > \frac{1}{q_{\mathsf{UA}}(\kappa)} \ .$$

Note that, in the definition of the weak proof-of-knowledge property, the knowledge extractor E_{UA} is required to run in probabilistic polynomial-time, while, on the other hand, the size of the witness for a particular instance $y = (M, x, t)$ may be superpolynomial in $|y|$; therefore, we can only require that the knowledge extractor is an "implicit representation" of a valid witness. Moreover, both E_{UA} and p' may depend on p, so that the proof of knowledge is "weak" in the sense that it does not imply the standard (or, "strong") proof of knowledge [BG93].

Barak and Goldreich prove the following theorem:

Theorem 10.7 ([BG02])**.** *If (standard) collision-resistant hashing schemes exist, then there exist (four-message, public coin) universal arguments.*

The weaker form of proof of knowledge that we consider does not insist that the knowledge extractor is an implicit representation of the witness, and thus we call this property *explicit* weak proof of knowledge:

Definition 10.8 (Explicit Weak Proof of Knowledge)**.** *For every two positive polynomials s_{UA} and p_{UA} there exist a positive polynomial q_{UA} and a probabilistic polynomial-time weak knowledge extractor E_{UA} such that for every family of s_{UA}-size prover circuits $\widetilde{P}_{\mathsf{UA}} = \{\widetilde{P}_{\mathsf{UA},\kappa}\}_{\kappa\in\mathbb{N}}$, for all sufficiently large $\kappa \in \mathbb{N}$, for every instance $y = (M, x, T) \in \{0,1\}^\kappa$ the following holds:*

Suppose that the prover circuit $\widetilde{P}_{\mathsf{UA},\kappa}$ convinces V_{UA} to accept y with probability greater than $p_{\mathsf{UA}}(k)^{-1}$ (taken over a random choice of internal randomness for V_{UA}).

Then, with probability greater than $q_{\mathsf{UA}}(k)^{-1}$ taken over a random choice of internal randomness r for E_{UA}, the weak knowledge extractor E_{UA}, with oracle access to the code of $\widetilde{P}_{\mathsf{UA},k}$ and on input $(y, 1^T)$, outputs a valid witness w for y,

In symbols:

$$\forall\, s_{\mathsf{UA}}\ \forall\, p_{\mathsf{UA}}\ \exists\, q_{\mathsf{UA}}\ \exists\, E_{\mathsf{UA}}\ \forall\, s_{\mathsf{UA}}\text{-}size\ \widetilde{P}_{\mathsf{UA}}\ \exists\, K\ \forall\, \kappa > K\ \forall\, y = (M, x, T) \in \{0,1\}^\kappa$$

if

$$\Pr\left[\langle \widetilde{P}_{\mathsf{UA},\kappa}, V_{\mathsf{UA}}\rangle(y) = 1\right] > \frac{1}{p_{\mathsf{UA}}(\kappa)}\ ,$$

then

$$\Pr_r\left[\, w \in R_{\mathcal{U}}(y) \,\middle|\, E_{\mathsf{UA}}^{\langle \widetilde{P}_{\mathsf{UA},\kappa}\rangle}(y, 1^T; r) = w \,\right] > \frac{1}{q_{\mathsf{UA}}(\kappa)}\ .$$

We thus define almost universal arguments as universal arguments where the proof of knowledge property is relaxed to that of Definition 10.8.

Definition 10.9. *An* **almost universal argument** *system $(P_{\mathsf{UA}}, V_{\mathsf{UA}})$ satisfies [BG02, Definition 2.1], except that the (implicit) weak proof-of-knowledge property is replaced by the explicit weak proof-of-knowledge property.*

As discussed already in Remark 10.1 in Section 2, we will not consider a "traditional" universal language (such as the universal language relative to random-access machines or relative to Turing machines), but we will concentrate on SuccinctACSPs because the properties we shall prove will remain "invariant" under appropriate Levin reductions. In such a case, instances consist of triples $y = (\mathsf{par}_{\text{sACSP}}, x, 1^t)$, where $\mathsf{par}_{\text{sACSP}}$ is a choice of parameters for SuccinctACSP and $(x, 1^t) \in \text{SuccinctACSP}(\mathsf{par}_{\text{sACSP}})$. Throughout, we fix a choice of $\mathsf{par}_{\text{sACSP}}$, so that it will suffice to mention the pair $(x, 1^t)$.

10.2 Relatively-Efficient Oracle Construction

The first property considered by Barak and Goldreich [BG02, Definiton 3.2, first item] for a PCP verifier is the existence of an efficient prover that is able to produce accepting PCP oracles whenever they exist:

Definition 10.10 (Relatively-Efficient Oracle Construction)**.** *A PCP verifier V has the* **relatively-efficient oracle construction property** *if there exists a polynomial-time algorithm P such that, on input any $(x, w) \in R$, algorithm P outputs an oracle π_x that makes V always accept.*

We remark the PCP verifier V_{ACSP} does satisfy the definition above:

Claim 10.11. *The PCP verifier V_{ACSP} has the relatively-efficient oracle construction property for the relation induced by the language* SuccinctACSP*.*

Proof. From Theorem 8.1 we know that the PCP prover P_{ACSP} does run in polynomial time. □

Clearly, this first property is the easiest property to establish because it is natural to most PCP constructions and, in our specific case, comes "for free" from our previous discussions (indeed, we have worked hard to make the PCP prover not only efficient but also fast!).

10.3 Non-Adaptive Verifier

The second property considered by Barak and Goldreich [BG02, Definiton 3.2, second item] for a PCP verifier is the non-adaptivity of the queries:

Definition 10.12 (Non-Adaptive PCP Verifier)**.** *A PCP verifier V is* **non-adaptive** *if its queries are based only on the input and its internal coin tosses, independently of the answers given to previous queries. That is, V can be decomposed into a pair of algorithms Q and D such that on input x and a random tape r, the verifier makes the query sequence $Q(x, r, 1), \ldots, Q(x, r, p(|x|))$ obtains the answers $b_1, \ldots, b_{p(|x|)}$, and decides according to $D(x, r, b_1 \cdots b_{p(|x|)})$, where p is some fixed polynomial.*

We prove that the PCP verifier V_{ACSP} does satisfy the definition above. While, at high level, this may be easy to see by inspection of the construction of V_{ACSP}, we believe it necessary to spell out in careful detail the proof that this is the case, because the "decomposition" of the verifier V_{ACSP} into a query algorithm and a decision algorithm is needed if one is interested in *constructing universal arguments* (by writing an implementation for them).

Claim 10.13. *The PCP verifier V_{ACSP} is a non-adaptive PCP verifier.*

The construction of V_{ACSP} is quite complicated, so we will have to proceed one step at a time. To begin with, as V_{ACSP} is constructed using (both standard and strong) PCPP verifiers, we also need to specify what we mean by a non-adaptive PCPP verifier:

Definition 10.14 (Non-Adaptive PCPP Verifier)**.** *A PCPP verifier V is* **non-adaptive** *if its queries are based only on the explicit input and its internal coin tosses, independently of the answers given to previous queries. That is, V can be decomposed into a pair of algorithms Q and D such that on explicit input x and a random tape r, the verifier makes the query sequence $Q(x, r, 1), \ldots, Q(x, r, p(|x|))$ obtains the answers $b_1, \ldots, b_{p(|x|)}$, and decides according to $D(x, r, b_1 \cdots b_{p(|x|)})$, where p is some fixed polynomial.*

Our proof will be "bottom up". Also, throughout, we fix a specific choice of parameters $(\eta, \kappa_0, \gamma, \mu)$, which parametrize V_{ACSP}. Finally, we will not carry out a time complexity analysis for the "non-adaptively decomposed" verifiers, as a detailed analysis for the "non-decomposed" verifiers was already carried out in Section B.

Remark 10.15. Note that the alphabet of proof oracles (as well as implicit inputs) are elements of a finite field $\mathbb{F}_{2^\ell}$, and not bits. Thus, our descriptions of Q and D will reason about indexing into strings of such field elements. The departure from binary oracles is mostly inconsequential, with the exception of a minor note that will come up later in the proof of almost universal arguments. (See Remark 10.41.)

10.3.1 Non-Adaptivity of $V_{\text{RS},=}$

Lemma 10.16. *$V_{\text{RS},=}$ is a non-adaptive (strong) PCPP verifier.*

Proof. The explicit input of $V_{\text{RS},=}$ is $(I_\ell, \mathcal{B}_L)$, where I_ℓ is an irreducible polynomial over $\mathbb{F}_2$ of degree ℓ with root x, which induces the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$, and $\mathcal{B}_L = (a_1, \ldots, a_\kappa)$ is a basis of a κ-dimensional linear subset $L \subseteq \mathbb{F}_{2^\ell}$; the implicit input of $V_{\text{RS},=}$ is a function $p\colon L \to \mathbb{F}_{2^\ell}$. The proof of proximity π of $V_{\text{RS},=}$ is parsed as a pair (f, Π), where f is a bivariate function over a subset of $\mathbb{F}_{2^\ell} \times \mathbb{F}_{2^\ell}$ and Π is a sequence of proofs of proximity for Reed-Solomon codes over (smaller) linear spaces. Finally, see Lemma B.1 for $\mathsf{query}_{\text{RS},=}(\ell, \kappa)$, $\mathsf{rand}_{\text{RS},=}(\ell, \kappa)$, and $\mathsf{length}_{\text{RS},=}(\ell, \kappa)$.

The algorithm $V_{\text{RS},=}$ is recursive, and all its queries to the implicit input and proof of proximity are made during its "base case", which occurs when $\dim(L) \leq \kappa_0$; the base case queries every value of the function found in the implicit input, thus obtaining a codeword $\alpha_1 \cdots \alpha_{2^\kappa}$, and then directly verifies whether $\alpha_1 \cdots \alpha_{2^\kappa}$ is in $\text{RS}(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1)$ or not (and accepts or rejects accordingly), where $d_{\kappa,\eta} := |L|/2^\eta - 1 = 2^{k-\eta} - 1$; since $\dim(L) \leq \kappa_0$, there are at most 2^{κ_0} values to query.

Therefore, at high level, the query algorithm $Q_{\text{RS},=}$ will be the algorithm that produces all the queries of the base case, while the decision algorithm $D_{\text{RS},=}$ will be the logic used by the base case to accept or reject, when given the answers to the (at most 2^{κ_0}) queries.

However, in the case of $Q_{\text{RS},=}$, the recursive nature of $V_{\text{RS},=}$ makes things slightly more complicated, because the implicit inputs used by recursive calls are "simulated" by the callers; thus, we must recursively "translate" queries, starting from the base case, to understand what "real" queries are made by the top-level algorithm (whose implicit input is not simulated by any other procedure), thus allowing us to write down $Q_{\text{RS},=}$. (Note that, unlike implicit inputs, proofs of proximity for recursive calls are already contained in the proof of proximity of the caller, and thus have no need to be simulated; consequently, we do not have to worry about translating queries to them.)

Similarly, also in the case of $D_{\text{RS},=}$, the recursive nature of $V_{\text{RS},=}$ makes things slightly more complicated, because the interpolation performed by $D_{\text{RS},=}$ (as part of its verification test) needs to know which linear subset of $\mathbb{F}_{2^\ell}$ is the domain of the function contained in the implicit input; this information depends on which recursive calls occurred starting from the top-level algorithm all the way down to the base case.

We now proceed to a detailed description of the algorithm $Q_{\text{RS},=}$, and then for the algorithm $D_{\text{RS},=}$.

The "hard work" (mainly, bookkeeping) in $Q_{\text{RS},=}$ is carried out by an iterative procedure that we call TRANSLATE; on input an irreducible polynomial I_ℓ of degree ℓ, a basis $\mathcal{B}_L$ for a

κ-dimensional linear subset $L \subseteq \mathbb{F}_{2^\ell}$, a random string $r \in \{0,1\}^{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$ for the verifier, and an index $i \in \{1,\ldots,\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)\}$, the procedure TRANSLATE outputs a triple $(\mathtt{text}, \mathsf{elt}, j)$, where $j \in \{1,\ldots,|L| + \mathsf{length}_{\mathrm{RS},=}(\ell,\kappa)\}$ is an index into the concatenation of the implicit input and proximity proof (which is a string of field elements) corresponding to the i-th query performed by $V_{\mathrm{RS},=}$ on input $(I_\ell, \mathcal{B}_L)$ and random string r; the other two components, $\mathtt{text} \in \{\text{“}\mathtt{implicit\text{-}input}\text{”}, \text{“}\mathtt{proximity\text{-}proof}\text{”}\}$ and $\mathsf{elt} \in L \cup \{S \cup T\}$, are additional information used by the iterative procedure.

Thus, the algorithm $Q_{\mathrm{RS},=}$ is as follows:

$Q_{\mathrm{RS},=}\Big((I_\ell, \mathcal{B}_L), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i\Big) \equiv$

1. Compute $(\mathtt{text}, \mathsf{elt}, j) :=$ TRANSLATE$(I_\ell, \mathcal{B}_L, r, i)$.
2. Output j.

The algorithm for TRANSLATE is the simple strategy that keeps track of index translation (cf. index translation in [BSS08, Section 6.3]) and indexes into the concatenation of the implicit input and proximity proof:

TRANSLATE$\Big(I_\ell, \mathcal{B}_L, r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i\Big) \equiv$

1. If $\kappa \le \kappa_0$, then do the following:
   (a) $\alpha_i :=$ GETELTWITHINDEX$(I_\ell, \mathcal{B}_L, i)$;
   (b) $\mathtt{text} :=$ “$\mathtt{implicit\text{-}input}$”;
   (c) $\mathsf{elt} := \alpha_i$;
   (d) output $(\mathtt{text}, \mathsf{elt}, i)$.
2. If $\kappa > \kappa_0$, then do the following:
   (a) $\mathcal{B}_{L_0} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma})$;
   (b) $\mathcal{B}_{L'_0} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu})$;
   (c) $[Z_{L_0}]^{\mathsf{A}} :=$ FINDSUBSPPOLY$(I_\ell, \mathcal{B}_{L_0})$;
   (d) if $r_1 = 0$, then do row test translation:
      i. $m := 1 + \lceil \kappa/2 \rceil + \gamma$;
      ii. $\mathcal{B}_{L_1} := (a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_\kappa)$;
      iii. $\beta :=$ GETRANDELT$(\mathcal{B}_{L_1}; r_2 \cdots r_{1 + \lceil \kappa/2 \rceil + \gamma})$;
      iv. $\beta' := [Z_{L_0}]^{\mathsf{A}}(\beta)$;
      v. $\iota_\beta :=$ GETINDEXOFELT$(I_\ell, \mathcal{B}_{L_1}, \beta)$;
      vi. if $\beta \in L'_0$, then $\mathcal{B}_{L_\beta} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1})$;
      vii. if $\beta \notin L'_0$, then $\mathcal{B}_{L_\beta} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}, \beta)$;
      viii. $(\mathtt{text}', \mathsf{elt}', \iota) :=$ TRANSLATE$(I_\ell, \mathcal{B}_{L_\beta}, r_{m+1} \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i)$;
      ix. if $\mathtt{text}' =$ “$\mathtt{implicit\text{-}input}$”, then:
         A. parse elt' as an element $\alpha \in L_\beta$;
         B. $\alpha' := [Z_{L_0}]^{\mathsf{A}}(\alpha)$;
         C. if $\alpha' = \beta'$, then:
            – $\mathtt{text} :=$ “$\mathtt{implicit\text{-}input}$”;
            – $\mathsf{elt} := \alpha$;
            – $i :=$ GETINDEXOFELT$(I_\ell, \mathcal{B}_L, \alpha)$;
            – output $(\mathtt{text}, \mathsf{elt}, i)$;
         D. if $\alpha' \neq \beta'$, then:
            – $\mathtt{text} :=$ “$\mathtt{proximity\text{-}proof}$”;
            – $\mathsf{elt} := (\alpha, \beta')$;
            – $i := |L| + |L_\beta| \cdot (\iota_\beta - 1) + \iota$;
            – output $(\mathtt{text}, \mathsf{elt}, i)$;
      x. if $\mathtt{text}' =$ “$\mathtt{proximity\text{-}proof}$”, then:
         A. $\mathtt{text} := \mathtt{text}'$;

B. $\mathsf{elt} := \mathsf{elt}'$;
C. $i = (\iota - |L_\beta| + |L|) + |L_\beta| \cdot |L_1| + (\iota_\beta - 1) \cdot \mathsf{length}_{\mathrm{RS},=}(\dim L_\beta)$;
D. output $(\mathtt{text}, \alpha, i)$;

(e) if $r_1 = 1$, then do column test translation:
i. $m := 1 + \lfloor \kappa/2 \rfloor - \gamma + \mu$;
ii. $\alpha := \textsc{GetRandElt}(\mathcal{B}_{L_0'}; r_2 \cdots r_{1+\lfloor \kappa/2 \rfloor - \gamma + \mu})$;
iii. $\alpha' := [Z_{L_0}]^{\mathsf{A}}(\alpha)$;
iv. $\iota_\alpha := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_{L_0'}, \alpha)$;
v. $\mathcal{B}_{Z_{L_0}(L_1)} := ([Z_{L_0}]^{\mathsf{A}}(a_{\lfloor \kappa/2 \rfloor - \gamma + 1}), \ldots, [Z_{L_0}]^{\mathsf{A}}(a_\kappa))$;
vi. $(\mathtt{text}', \mathsf{elt}', \iota) := \textsc{Translate}(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, r_{m+1} \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i)$;
vii. if $\mathtt{text}' =$ "`implicit-input`", then:
A. parse elt' as an element $\beta' \in Z_{L_0}(L_1)$;
B. if $\alpha' = \beta'$, then:
– $\mathtt{text} :=$ "`implicit-input`";
– $\mathsf{elt} := \alpha$;
– $i := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_L, \alpha)$;
– output $(\mathtt{text}, \mathsf{elt}, i)$;
C. if $\alpha' \neq \beta'$, then:
– $\mathtt{text} :=$ "`proximity-proof`";
– $\mathsf{elt} := (\alpha, \beta')$;
– $i := |L| + |L_\beta| \cdot (\iota - 1) + \iota_\alpha$;
– output $(\mathtt{text}, \mathsf{elt}, i)$;
viii. if $\mathtt{text}' =$ "`proximity-proof`", then:
A. $\mathtt{text} := \mathtt{text}'$;
B. $\mathsf{elt} := \mathsf{elt}'$;
C. $i = (\iota - |Z_{L_0}(L_1)| + |L|) + |L_\beta| \cdot |L_1| + |L_1| \cdot \mathsf{length}_{\mathrm{RS},=}(\dim L_\beta) + (\iota_\alpha - 1) \cdot \mathsf{length}_{\mathrm{RS},=}(\dim Z_{L_0}(L_1))$;
D. output $(\mathtt{text}, \alpha, i)$.

The algorithm for $D_{\mathrm{RS},=}$ is somewhat simpler than the one for $Q_{\mathrm{RS},=}$, because we only need to keep track of the domain of the function in the implicit input.

$$D_{\mathrm{RS},=}\Big((I_\ell, \mathcal{B}_L), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha_1 \cdots \alpha_{2^\kappa}\Big) \equiv$$

1. If $\kappa \leq \kappa_0$, then:
(a) $d_{\kappa,\eta} = 2^\kappa / 2^\eta - 1$;
(b) $P := \textsc{SubspaceInterp}(I_\ell, \mathcal{B}_L, (\alpha_1, \ldots, \alpha_{2^\kappa}))$
(c) $\tilde{d} := \deg(P)$
(d) if $\tilde{d} \leq d_{\kappa,\eta}$, output 1, else output 0
2. If $\kappa > \kappa_0$, then:
(a) if $r_1 = 0$, then do row test recursion:
i. $m := 1 + \lceil \kappa/2 \rceil + \gamma$;
ii. $\mathcal{B}_{L_1} := (a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_\kappa)$;
iii. $\beta := \textsc{GetRandElt}(\mathcal{B}_{L_1}; r_2 \cdots r_{1+\lceil \kappa/2 \rceil + \gamma})$;
iv. if $\beta \in L_0'$, then $\mathcal{B}_{L_\beta} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1})$;
v. if $\beta \notin L_0'$, then $\mathcal{B}_{L_\beta} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}, \beta)$;
vi. output $D_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_{L_\beta}), r_{m+1} \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha_1 \cdots \alpha_{2^\kappa}\big)$;
(b) if $r_1 = 1$, then do column test recursion:
i. $m := 1 + \lfloor \kappa/2 \rfloor - \gamma + \mu$;
ii. $\mathcal{B}_{L_0} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma})$;
iii. $[Z_{L_0}]^{\mathsf{A}} := \textsc{FindSubspPoly}(I_\ell, \mathcal{B}_{L_0})$;

iv. $\mathcal{B}_{L'_0} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu})$;
v. $\alpha := \textsc{GetRandElt}(\mathcal{B}_{L'_0}; r_2 \cdots r_{1+\lfloor \kappa/2 \rfloor - \gamma + \mu})$;
vi. $\mathcal{B}_{Z_{L_0}(L_1)} := ([Z_{L_0}]^{\mathsf{A}}(a_{\lfloor \kappa/2 \rfloor - \gamma + 1}), \ldots, [Z_{L_0}]^{\mathsf{A}}(a_\kappa))$;
vii. output $D_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_{q(L_1)}), r_{m+1} \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha_1 \cdots \alpha_{2^\kappa}\big)$.

The correctness of both $Q_{\mathrm{RS},=}$ and $D_{\mathrm{RS},=}$ easily follows by inspection of the algorithms and the above discussion. □

10.3.2 Non-Adaptivity of $V_{\mathrm{RS},<}$

Lemma 10.17. *$V_{\mathrm{RS},<}$ is a non-adaptive (strong) PCPP verifier.*

Proof. The explicit input of $V_{\mathrm{RS},<}$ is $(I_\ell, \mathcal{B}_S, d)$, where I_ℓ is an irreducible polynomial over $\mathbb{F}_2$ of degree ℓ with root x, which induces the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$, $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ is a basis of a κ-dimensional linear subset $S \subseteq \mathbb{F}_{2^\ell}$, and d is a positive integer that is less than $d_{\kappa,\eta} := |S|/2^\eta - 1 = 2^{\kappa-\eta} - 1$; the implicit input of $V_{\mathrm{RS},<}$ is a function $p\colon S \to \mathbb{F}_{2^\ell}$. The proof of proximity π of $V_{\mathrm{RS},<}$ is parsed as a pair (π_1, π_2), where both π_1 and π_2 are proofs of proximity to $\mathrm{RS}(\mathbb{F}_{2^\ell}, S, d_{\kappa,\eta})$. Finally, see Lemma B.2 for $\mathsf{query}_{\mathrm{RS},<}(\ell, \kappa, d)$, $\mathsf{rand}_{\mathrm{RS},<}(\ell, \kappa, d)$, and $\mathsf{length}_{\mathrm{RS},<}(\ell, \kappa, d)$.

The algorithm $V_{\mathrm{RS},<}$ simply calls $V_{\mathrm{RS},=}$ twice with different inputs, and hence makes at most $\mathsf{query}_{\mathrm{RS},<}(\ell, \kappa, d) := 2 \cdot \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)$ queries. Thus, the algorithm $Q_{\mathrm{RS},<}$ calls $Q_{\mathrm{RS},=}$ and then, depending on whether the query was in the first set of $\mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)$ queries or in the second set of $\mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)$ queries, does not shift or shifts by the right amount the index output by $Q_{\mathrm{RS},=}$.[19]

Thus, the algorithm $Q_{\mathrm{RS},<}$ is defined as follows:

$Q_{\mathrm{RS},<}\Big((I_\ell, \mathcal{B}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d)}, i\Big) \equiv$

1. If $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)\}$, then:
   (a) $r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$;
   (b) $j := Q_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S), r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i)$;
   (c) output j.
2. If $\iota \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)\}$, where $\iota := i - \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)$, then:
   (a) $r''_1 \cdots r''_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$;
   (b) $j_0 := Q_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S), r''_1 \cdots r''_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \iota)$;
   (c) if $0 < j_0 \le |S|$, then $j := j_0$;
   (d) if $|S| < j_0 \le |S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa)$, then $j := \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa) + j_0$;
   (e) output j.

(Note that it just so happens that $Q_{\mathrm{RS},=}$ is independent of d.) Note that, in Step 2, we need to "shift left" the query by $\mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)$ before feeding it to $Q_{\mathrm{RS},=}$ and, after invoking $Q_{\mathrm{RS},=}$, we may need to "shift right" its output index, depending if it is to the explicit input or the proximity proof.

[19] Note that $V_{\mathrm{RS},<}$ uses its own implicit input as the implicit input to the first call of $V_{\mathrm{RS},=}$, while it simulates the implicit input to the second call of $V_{\mathrm{RS},=}$; it turns out that this simulation for the second call of $V_{\mathrm{RS},=}$ does not require us to modify the indices returned by $Q_{\mathrm{RS},=}$ for queries in the second set of $\mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)$ queries (besides, shifting the indices by the right amount, that is). (Indeed, simulating a query to $p'(\alpha) = p(\alpha) \cdot Q(\alpha)$ by querying $p(\alpha)$ and multiplying the result by $Q(\alpha)$ has the obvious property that the values of p' and those of p appear in the same order in the evaluation tables of either.)

The corresponding algorithm $D_{\mathrm{RS},=}$ is defined as follows:

$D_{\mathrm{RS},<}\Big((I_\ell, \mathcal{B}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d)}\Big) \equiv$

1. Parse $\alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d)}$ as $\alpha'_1 \cdots \alpha'_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)} \alpha''_1 \cdots \alpha''_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)}$.
2. $r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$.
3. $b_1 := D_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S), r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha'_1 \cdots \alpha'_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)})$,
4. $r''_1 \cdots r''_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$.
5. $d_{\kappa,\eta} := |S|/2^\eta - 1$.
6. For $i = 1, \ldots, \mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)$, do the following:
   (a) $j_0 := Q_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S), r''_1 \cdots r''_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i)$;
   (b) if $0 < j_0 \le |S|$, then:
      i. $\gamma_{j_0} := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, j_0)$;
      ii. reset α''_i to the new value $\gamma_{j_0}^{d_{\kappa,\eta}-d} \alpha''_i$.
7. $b_2 := D_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S), r''_1 \cdots r''_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha''_1 \cdots \alpha''_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)})$.
8. $b := b_1 \wedge b_2$.
9. Output b.

(Note that it just so happens that $D_{\mathrm{RS},<}$ is independent of d.) Note that $D_{\mathrm{RS},<}$ is given as input two sets of query answers $\alpha'_1 \cdots \alpha'_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)}$ and $\alpha''_1 \cdots \alpha''_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)}$, and invokes $D_{\mathrm{RS},=}$ once on each; also, before invoking $D_{\mathrm{RS},=}$ on the second set $\alpha''_1 \cdots \alpha''_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)}$, $D_{\mathrm{RS},<}$ has to first multiply the values to simulate answers from the function $p \cdot x^{d_{\kappa,\eta}-d}$ instead of p.

The correctness of both $Q_{\mathrm{RS},<}$ and $D_{\mathrm{RS},<}$ easily follows by inspection of the algorithms and the above discussion, as well as Lemma 10.16. □

10.3.3 Non-Adaptivity of $V_{\mathrm{RS},>}$

Lemma 10.18. *$V_{\mathrm{RS},>}$ is a non-adaptive (strong) PCPP verifier.*

Proof. The explicit input of $V_{\mathrm{RS},>}$ is $(I_\ell, \mathcal{B}_S, d)$, where I_ℓ is an irreducible polynomial over $\mathbb{F}_2$ of degree ℓ with root x, which induces the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$, $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ is a basis of a κ-dimensional linear subset S of $\mathbb{F}_{2^\ell}$, and d is a positive integer that is greater than $d_{\kappa,\eta} := |S|/2^\eta - 1 = 2^{\kappa-\eta} - 1$; the implicit input of $V_{\mathrm{RS},>}$ is a function $p\colon S \to \mathbb{F}_{2^\ell}$. The proof of proximity π of $V_{\mathrm{RS},>}$ is parsed as a 2^η-tuple of pairs $((p_0, \pi_0), \ldots, (p_{2^\eta-1}, \pi_{2^\eta-1}))$, where $p_0, \ldots, p_{2^\eta-1}$ are functions from S to $\mathbb{F}_{2^\ell}$ and $\pi_0, \ldots, \pi_{2^\eta-1}$ are proofs of proximity to $\mathrm{RS}(\mathbb{F}_{2^\ell}, S, d_0), \ldots, \mathrm{RS}(\mathbb{F}_{2^\ell}, S, d_{2^\eta-1})$, where $d_0, \ldots, d_{2^\eta-1}$ are the 2^η unique non-negative integers such that a polynomial $P(x)$ of degree $d < |S| = 2^\eta(d_{\kappa,\eta}+1)$ can be written as a sum $\sum_{i=0}^{2^\eta-1} x^{i(d_{\kappa,\eta}+1)} P_i(x)$ with $\deg P_i = d_i \le d_{\kappa,\eta}$. Finally, see Lemma B.3 for $\mathsf{query}_{\mathrm{RS},>}(\ell,\kappa,d)$, $\mathsf{rand}_{\mathrm{RS},>}(\ell,\kappa,d)$, and $\mathsf{length}_{\mathrm{RS},>}(\ell,\kappa,d)$.

Let $m_{\kappa,\eta,d} := \lfloor (d+1)/(d_{\kappa,\eta}+1) \rfloor$. The algorithm $V_{\mathrm{RS},>}$ does some "real" verification only in the case $d < |S| = 2^\kappa$, for otherwise it can always accept because $\mathrm{RS}(\mathbb{F}_{2^\ell}, S, d)$ includes all the codewords (of length $|S|$). Hence, $d_{\kappa,\eta} < d < |S| = 2^\eta(d_{\kappa,\eta}+1)$, and thus $m_{\kappa,\eta,d} \in \{0, \ldots, 2^\eta\}$. The algorithm $V_{\mathrm{RS},>}$ makes $m_{\kappa,\eta,d}$ calls to $V_{\mathrm{RS},=}$ and, in case $m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1) < (d+1)$, also one call to $V_{\mathrm{RS},<}$; after that, it performs a consistency test by querying at one value the implicit input and at one value each of the functions $p_0, \ldots, p_{2^\eta}$ in the proximity proof (for a total of $1 + 2^\eta$ additional queries).

Thus, the algorithm $Q_{\mathrm{RS},>}$ is defined as follows:

$Q_{\mathrm{RS},>}\Big((I_\ell, \mathcal{B}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},>}(\ell,\kappa,d)}, i\Big) \equiv$

1. If $d \geq 2^\kappa$, then output $\perp$. (Because the decision algorithm $D_{\mathrm{RS},>}$ will accept without examining any answer to any query.)
2. If $d < 2^\kappa$, then:
   (a) $d_{\kappa,\eta} := |S|/2^\eta - 1$;
   (b) $m_{\kappa,\eta,d} := \lfloor (d+1)/(d_{\kappa,\eta}+1) \rfloor$;
   (c) $d_{\kappa,\eta,d} := d - m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1)$;
   (d) if $i \in \{1, \ldots, m_{\kappa,\eta,d} \cdot \mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)\}$, then:
      i. $z := \lfloor (i-1)/\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa) \rfloor$;
      ii. $r_1^{(z)} \cdots r^{(z)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$;
      iii. $i' := i - z \cdot \mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)$;
      iv. $j_0 := Q_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S), r_1^{(z)} \cdots r^{(z)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i')$;
      v. $j := |S| + z \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell,\kappa)) + j_0$;
      vi. output j;
   (e) if $m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1) < d+1$ and $i' \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d)\}$, where $i' := i - m_{\kappa,\eta,d} \cdot \mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)$, then:
      i. $r_1^{(m_{\kappa,\eta,d}+1)} \cdots r^{(m_{\kappa,\eta,d}+1)}_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})}$;
      ii. $j_0 := Q_{\mathrm{RS},<}((I_\ell, \mathcal{B}_S, d_{\kappa,\eta,d}), r_1^{(m_{\kappa,\eta,d}+1)} \cdots r^{(m_{\kappa,\eta,d}+1)}_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})}, i')$;
      iii. $j := |S| + m_{\kappa,\eta,d} \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell,\kappa)) + j_0$;
      iv. output j;
   (f) if $m \cdot (d_{\kappa,\eta}+1) = d+1$ and $i_0 \in \{1, \ldots, 1+2^\eta\}$, where $i_0 := i - m_{\kappa,\eta,d} \cdot \mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)$, then:
      i. $r_1^{(\gamma)} \cdots r_\kappa^{(\gamma)} := r_1 \cdots r_\kappa$;
      ii. $\gamma := \textsc{GetRandElt}(\mathcal{B}_S; r_1^{(\gamma)} \cdots r_\kappa^{(\gamma)})$;
      iii. $\iota_\gamma := \textsc{GetIndexOfElt}(I_t, \mathcal{B}_S, \gamma)$;
      iv. if $i_0 = 1$, then $j := \iota_\gamma$;
      v. if $1 < i_0 \leq (m_{\kappa,\eta,d}+1)$, then $j := |S| + (i_0 - 2) \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell,\kappa)) + \iota_\gamma$;
      vi. if $(m_{\kappa,\eta,d}+1) < i_0 \leq 2^\eta + 1$, then $j := \perp$;
      vii. output j;
   (g) if $m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1) < d+1$ and $i_0 \in \{1, \ldots, 1+2^\eta\}$, where $i_0 := i - (m_{\kappa,\eta,d} \cdot \mathsf{query}_{\mathrm{RS},=}(\ell,\kappa) + \mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d}))$, then:
      i. $r_1^{(\gamma)} \cdots r_\kappa^{(\gamma)} := r_1 \cdots r_\kappa$;
      ii. $\gamma := \textsc{GetRandElt}(\mathcal{B}_S; r_1^{(\gamma)} \cdots r_\kappa^{(\gamma)})$;
      iii. $\iota_\gamma := \textsc{GetIndexOfElt}(I_t, \mathcal{B}_S, \gamma)$;
      iv. if $i_0 = 1$, then $j := \iota_\gamma$;
      v. if $1 < i_0 \leq (m_{\kappa,\eta,d}+1)$, then $j := |S| + (i_0 - 2) \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell,\kappa)) + \iota_\gamma$;
      vi. if $i = m_{\kappa,\eta,d} + 2$, then $j := |S| + m_{\kappa,\eta,d} \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell,\kappa)) + \iota_\gamma$;
      vii. if $(m_{\kappa,\eta,d}+2) < i_0 \leq 2^\eta + 1$, then $j := \perp$;
      viii. output j.

The corresponding algorithm $D_{\mathrm{RS},>}$ is defined as follows:

$D_{\mathrm{RS},>}\Big((I_\ell, \mathcal{B}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},>}(\ell,\kappa,d)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS},>}(\ell,\kappa,d)}\Big) \equiv$

1. If $d \geq 2^\kappa$, then output 1.

2. If $d < 2^\kappa$, then:
   (a) $r_1^{(\gamma)} \cdots r_\kappa^{(\gamma)} := r_1 \cdots r_\kappa$;
   (b) $\gamma := \textsc{GetRandElt}(\mathcal{B}_S; r_1^{(\gamma)} \cdots r_\kappa^{(\gamma)})$;
   (c) $m_{\kappa,\eta,d} := \lfloor (d+1)/(d_{\kappa,\eta}+1) \rfloor$;
   (d) $d_{\kappa,\eta,d} := d - m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1)$;
   (e) if $m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1) < d+1$, then:
      i. parse $\alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS},>}(\ell,\kappa,d)}$ as

$$\alpha_1^{(0)} \cdots \alpha^{(0)}_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)} \cdots \alpha_1^{(m_{\kappa,\eta,d}-1)} \cdots \alpha^{(m_{\kappa,\eta,d}-1)}_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)} \alpha_1^{(m_{\kappa,\eta,d})} \cdots \alpha^{(m_{\kappa,\eta,d})}_{\mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})} v v_0 \cdots v_{2^\eta - 1} \ ;$$

      ii. for $i = 0, \ldots, m_{\kappa,\eta,d} - 1$, do the following:
         A. $r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$;
         B. $b_i := D_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S), r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha_1^{(i)} \cdots \alpha^{(i)}_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)})$;
      iii. $r_1^{(m_{\kappa,\eta,d})} \cdots r^{(m_{\kappa,\eta,d})}_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})}$;
      iv. $b_{m_{\kappa,\eta,d}} := D_{\mathrm{RS},<}((I_\ell, \mathcal{B}_S), r_1^{(m_{\kappa,\eta,d})} \cdots r^{(m_{\kappa,\eta,d})}_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})}, \alpha_1^{(m_{\kappa,\eta,d})} \cdots \alpha^{(m_{\kappa,\eta,d})}_{\mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})})$;
      v. $b_{\mathsf{consist}} := (v \stackrel{?}{=} \sum_{i=0}^{2^\eta - 1} \gamma^{i(d_{\kappa,\eta}+1)} v_i)$;
      vi. $b := b_{\mathsf{consist}} \wedge b_0 \wedge \cdots \wedge b_{m_{\kappa,\eta,d}-1} \wedge b_{m_{\kappa,\eta,d}}$;
      vii. output b;
   (f) if $m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1) = d+1$, then:
      i. parse $\alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS},>}(\ell,\kappa,d)}$ as

$$\alpha_1^{(0)} \cdots \alpha^{(0)}_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)} \cdots \alpha_1^{(m_{\kappa,\eta,d}-1)} \cdots \alpha^{(m_{\kappa,\eta,d}-1)}_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)} v v_0 \cdots v_{2^\eta - 1} \ ;$$

      ii. for $i = 0, \ldots, m_{\kappa,\eta,d} - 1$, do
         A. $r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$;
         B. $b_i := D_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S), r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha_1^{(i)} \cdots \alpha^{(i)}_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)})$;
      iii. $b_{\mathsf{consist}} := (v \stackrel{?}{=} \sum_{i=0}^{2^\eta - 1} \gamma^{i(d_{\kappa,\eta}+1)} v_i)$;
      iv. $b := b_{\mathsf{consist}} \wedge b_0 \wedge \cdots \wedge b_{m-1}$;
      v. output b.

The correctness of both $Q_{\mathrm{RS},>}$ and $D_{\mathrm{RS},>}$ easily follows by inspection of the algorithms and the above discussion, as well as Lemma 10.16 and Lemma 10.17. □

10.3.4 Non-Adaptivity of V_{RS}

Lemma 10.19. *V_{RS} is a non-adaptive (strong) PCPP verifier.*

Proof. The explicit input of V_{RS} is $(I_\ell, \mathcal{B}_S, d)$, where I_ℓ is an irreducible polynomial over $\mathbb{F}_2$ of degree ℓ with root x, which induces the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$, $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ is a basis of a κ-dimensional linear subset $S \subseteq \mathbb{F}_{2^\ell}$, and d is a positive integer; the implicit input of V_{RS} is a function $p \colon S \to \mathbb{F}_{2^\ell}$. The proof of proximity π of V_{RS} is parsed according to whether d is less than, equal to, or greater than $d_{\kappa,\eta} := |S|/2^\eta - 1 = 2^{\kappa-\eta} - 1$. Finally, see Lemma B.4 for $\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)$, $\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)$, and $\mathsf{length}_{\mathrm{RS}}(\ell,\kappa,d)$.

The algorithm V_{RS} simply calls $V_{\mathrm{RS},<}$, $V_{\mathrm{RS},=}$, or $V_{\mathrm{RS},>}$ according to the three cases for d with respect to $d_{\kappa,\eta}$. (Unless $\kappa \leq \eta$, in which case V_{RS} directly tests the degree.) The algorithms for Q_{RS} and D_{RS} therefore follow immediately.

Thus, the algorithm Q_{RS} is defined as follows:

$$Q_{\mathrm{RS}}\Big((I_\ell, \mathcal{B}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)}, i\Big) \equiv$$

1. If $\kappa \leq \eta$:
   (a) Output $j := i$.
2. If $\kappa > \eta$:
   (a) Set $d_{\kappa,\eta} := |S|/2^\eta - 1$.
   (b) If $d < d_{\kappa,\eta}$, then output $j := Q_{\mathrm{RS},<}\big((I_\ell, \mathcal{B}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d)}, i\big)$.
   (c) If $d = d_{\kappa,\eta}$, then output $j := Q_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i\big)$.
   (d) If $d > d_{\kappa,\eta}$, then output $j := Q_{\mathrm{RS},>}\big((I_\ell, \mathcal{B}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},>}(\ell,\kappa,d)}, i\big)$.

The corresponding algorithm D_{RS} is defined as follows:

$$D_{\mathrm{RS}}\Big((I_\ell, \mathcal{B}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)}\Big) \equiv$$

1. If $\kappa \leq \eta$:
   (a) $P := \textsc{SubspaceInterp}(I_\ell, \mathcal{B}_S, (\alpha_1, \ldots, \alpha_{2^\kappa}))$
   (b) $\tilde{d} := \deg(P)$
   (c) if $\tilde{d} \leq d$, output 1, else output 0
2. If $\kappa > \eta$:
   (a) Set $d_{\kappa,\eta} := |S|/2^\eta - 1$.
   (b) If $d < d_{\kappa,\eta}$, then output $b := D_{\mathrm{RS},<}\big((I_\ell, \mathcal{B}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d)}\big)$.
   (c) If $d = d_{\kappa,\eta}$, then output $b := D_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)}\big)$.
   (d) If $d > d_{\kappa,\eta}$, then output $b := D_{\mathrm{RS},>}\big((I_\ell, \mathcal{B}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},>}(\ell,\kappa,d)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS},>}(\ell,\kappa,d)}\big)$.

The correctness of both Q_{RS} and D_{RS} easily follows by inspection of the algorithms and the above discussion, as well as Lemma 10.16, Lemma 10.17, and Lemma 10.18. □

10.3.5 Non-Adaptivity of V_{VRS}

Lemma 10.20. *V_{VRS} is a non-adaptive (strong) PCPP verifier.*

Proof. The explicit input of V_{VRS} is $(I_\ell, \mathcal{B}_S, \mathcal{B}_H, d)$, where I_ℓ is an irreducible polynomial over $\mathbb{F}_2$ of degree ℓ with root x, which induces the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$, $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ is a basis of a κ-dimensional linear subset $S \subseteq \mathbb{F}_{2^\ell}$, $\mathcal{B}_H = (b_1, \ldots, b_\lambda)$ is a basis of a λ-dimensional linear subset $H \subseteq \mathbb{F}_{2^\ell}$, and d is a positive integer; the implicit input of V_{VRS} is a function $p\colon S \to \mathbb{F}_{2^\ell}$. The proof of proximity π of V_{VRS} is parsed as a pair $(\tilde{p}, \tilde{\pi})$ where $\tilde{\pi}$ is a proof of proximity for the function $\tilde{p}$ to $\mathrm{RS}(\mathbb{F}_{2^\ell}, S, d - |H|)$. Finally, see Lemma B.5 for $\mathsf{query}_{\mathrm{VRS}}(\ell, \kappa, \lambda, d)$, $\mathsf{rand}_{\mathrm{VRS}}(\ell, \kappa, \lambda, d)$, and $\mathsf{length}_{\mathrm{VRS}}(\ell, \kappa, \lambda, d)$.

The algorithm V_{VRS} simply calls V_{RS} on input $(I_\ell, \mathcal{B}_S, d - |H|)$ (using the proximity proof $\pi = (\tilde{p}, \tilde{\pi})$ as the pair of oracles for V_{RS}), then makes two queries (one to the implicit input p, and one to the function $\tilde{p}$ in the proximity proof π), and performs a consistency test between the functions p and $\tilde{p}$.

Thus, the algorithm Q_{VRS} is defined as follows:

$Q_{\mathrm{VRS}}\Big((I_\ell, \mathcal{B}_S, \mathcal{B}_H, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}, i\Big) \equiv$

1. If $i \in \big\{1, \ldots, \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d)\big\}$, then:
   (a) $r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d-|H|)} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d-|H|)}$;
   (b) $j_{\mathsf{old}} := Q_{\mathrm{RS}}\big((I_\ell, \mathcal{B}_S, d - |H|), r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d-|H|)}, i\big)$;
   (c) $j := |S| + j_{\mathsf{old}}$;
   (d) output j.
2. If $i = \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d) + 1$, then:
   (a) $r_1^{(\alpha)} \cdots r_\kappa^{(\alpha)} := r_1 \cdots r_\kappa$;
   (b) $\alpha := \textsc{GetRandElt}(\mathcal{B}_S; r_1^{(\alpha)} \cdots r_\kappa^{(\alpha)})$;
   (c) $j_{\mathsf{old}} := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_S, \alpha)$;
   (d) $j := j_{\mathsf{old}}$;
   (e) output j.
3. If $i = \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d) + 2$, then:
   (a) $r_1^{(\alpha)} \cdots r_\kappa^{(\alpha)} := r_1 \cdots r_\kappa$;
   (b) $\alpha := \textsc{GetRandElt}(\mathcal{B}_S; r_1^{(\alpha)} \cdots r_\kappa^{(\alpha)})$;
   (c) $j_{\mathsf{old}} := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_S, \alpha)$;
   (d) $j := |S| + j_{\mathsf{old}}$;
   (e) output j.

The corresponding algorithm D_{VRS} is defined as follows:

$D_{\mathrm{VRS}}\Big((I_\ell, \mathcal{B}_S, \mathcal{B}_H, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}\Big) \equiv$

1. Parse $\alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}$ as $\alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d-|H|)} \omega \widetilde{\omega}$.
2. $r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d-|H|)} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d-|H|)}$.
3. $b_{\mathrm{RS}} := D_{\mathrm{RS}}\big((I_\ell, \mathcal{B}_S, d - |H|), r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d-|H|)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d-|H|)}\big)$.
4. $r_1^{(\alpha)} \cdots r_\kappa^{(\alpha)} := r_1 \cdots r_\kappa$.
5. $\alpha := \textsc{GetRandElt}(\mathcal{B}_S; r_1^{(\alpha)} \cdots r_\kappa^{(\alpha)})$.
6. $[Z_H]^{\mathsf{A}} := \textsc{FindSubspPoly}(I_\ell, \mathcal{B}_H)$;
7. $\tau := [Z_H]^{\mathsf{A}}(\alpha)$.
8. If $(\omega = \tau \cdot \widetilde{\omega})$, then $b_{\mathsf{consist}} := 1$ else $b_{\mathsf{consist}} := 0$.
9. $b := b_{\mathrm{RS}} \wedge b_{\mathsf{consist}}$.
10. Output b.

The correctness of both Q_{VRS} and D_{VRS} easily follows by inspection of the algorithms and the above discussion, as well as Lemma 10.19. □

10.3.6 Non-Adaptivity of V_{aRS}

Lemma 10.21. *V_{aRS} is a non-adaptive PCPP verifier.*

Proof. The algorithm V_{aRS} is (as the name suggests) an algorithm that amplifies the soundness of V_{RS}; we let $s_{\mathrm{RS}}(\delta, n)$ denote the soundness function of V_{RS}. With the exception of the addition of the proximity parameter δ and target constant soundness $s' \in [0, 1]$, V_{aRS} has the same explicit input, implicit input, and proof of proximity structure as V_{RS}; all of these were briefly recalled in Lemma 10.19. (Note that there is no prover algorithm that is "specialized" for V_{aRS},

because its corresponding prover is the same as the one for V_{RS}. Finally, see Lemma B.6 for $\mathsf{query}_{\mathrm{aRS}}(\ell,\kappa,d,\delta,s')$ and $\mathsf{rand}_{\mathrm{aRS}}(\ell,\kappa,d,\delta,s')$.

The algorithm V_{aRS} simply calls V_{RS} several times, on the same input but with different randomness, and then accepts if and only if all the calls to V_{RS} accept. Hence, the algorithms Q_{aRS} and D_{aRS} for V_{aRS} can easily be constructed using the algorithms Q_{RS} and D_{RS} for V_{RS}.

Specifically, the algorithm Q_{aRS} is defined as follows:

$$Q_{\mathrm{aRS}}\Big((I_\ell,\mathcal{B}_S,d,\delta,s'),r_1\cdots r_{\mathsf{rand}_{\mathrm{aRS}}(\ell,\kappa,d,\delta,s')},i\Big)\equiv$$

1. $m:=\left\lceil\frac{\log(1-s')}{\log(1-s_{\mathrm{RS}}(\delta,2^\kappa))}\right\rceil$.
2. Let $z\in\{1,\ldots,m\}$ be such that $i\in\big\{(z-1)\cdot\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)+1,\ldots,z\cdot\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)\big\}$.
3. $i_{\mathsf{new}}:=i-(z-1)\cdot\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)$.
4. $r_1^{(i_{\mathsf{new}})}\cdots r^{(i_{\mathsf{new}})}_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)}:=r_{(i_{\mathsf{new}}-1)\cdot\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)+1}\cdots r_{i_{\mathsf{new}}\cdot\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)}$.
5. $j:=Q_{\mathrm{RS}}\big((I_\ell,\mathcal{B}_S,d),r_1^{(i_{\mathsf{new}})}\cdots r^{(i_{\mathsf{new}})}_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)},i_{\mathsf{new}}\big)$.
6. Output j.

The corresponding algorithm D_{aRS} is defined as follows:

$$D_{\mathrm{aRS}}\Big((I_\ell,\mathcal{B}_S,d,\delta,s'),r_1\cdots r_{\mathsf{rand}_{\mathrm{aRS}}(\ell,\kappa,d,\delta,s')},\alpha_1\cdots\alpha_{\mathsf{query}_{\mathrm{aRS}}(\ell,\kappa,d,\delta,s')}\Big)\equiv$$

1. $m:=\left\lceil\frac{\log(1-s')}{\log(1-s_{\mathrm{RS}}(\delta,2^\kappa))}\right\rceil$.
2. Parse $\alpha_1\cdots\alpha_{\mathsf{query}_{\mathrm{aRS}}(\ell,\kappa,\delta,s')}$ as $\alpha_1^{(1)}\cdots\alpha^{(1)}_{\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)}\cdots\alpha_1^{(m)}\cdots\alpha^{(m)}_{\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)}$.
3. For $i=1,\ldots,m$, do the following:
   (a) $r_1^{(i)}\cdots r^{(i)}_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)}:=r_{(i-1)\cdot\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)+1}\cdots r_{i\cdot\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)}$;
   (b) $b_i:=D_{\mathrm{RS}}\big((I_\ell,\mathcal{B}_S,d),r_1^{(i)}\cdots r^{(i)}_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)},\alpha_1^{(i)}\cdots\alpha^{(i)}_{\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)}\big)$.
4. $b:=b_1\wedge\cdots\wedge b_m$.
5. Output b.

The correctness of both Q_{aRS} and D_{aRS} easily follows by inspection of the algorithms and the above discussion, as well as Lemma 10.19. □

Note that, in our explicit description of the algorithm V_{RS}, we have chosen to not worry about randomness efficiency, so that each run of the sequential repetition is performed using fresh randomness; this is in contrast to the randomness-efficient result given for this step in [BSS08, Proposition 2.9]. Of course, as was mentioned before (see Remark 6.2), the reason is that a randomness-efficient sampler would complicate the construction unnecessarily, given that in this work we are not worried about randomness efficiency. Of course, if one wanted to consider a randomness-efficient V_{aRS}, one could easily modify the above algorithms Q_{aRS} and D_{aRS} to account for that, by simply changing how the two algorithms give randomness to each sequential run.

10.3.7 Non-Adaptivity of V_{aVRS}

Lemma 10.22. *V_{aVRS} is a non-adaptive PCPP verifier.*

Proof. This proof is completely analogous to that of Lemma 10.21, but, for completeness, we write it in full. The algorithm V_{aVRS} is (as the name suggests) an algorithm that amplifies the soundness of V_{VRS}; we let $s_{\text{VRS}}(\delta, n)$ denote the soundness function of V_{RS}. With the exception of the addition of the proximity parameter δ and the target soundness s', V_{aVRS} has the same explicit input, implicit input, and proof of proximity structure as V_{VRS}; all of these were briefly recalled in Lemma 10.20. (Note that there is no prover algorithm that is "specialized" for V_{aVRS}, because its corresponding prover is the same as the one for V_{VRS}.) Finally, see Lemma B.7 for $\mathsf{query}_{\text{aVRS}}(\ell, \kappa, \lambda, d, \delta, s')$ and $\mathsf{rand}_{\text{aVRS}}(\ell, \kappa, \lambda, d, \delta, s')$.

The algorithm V_{aVRS} simply calls V_{VRS} several times, on the same input but with different randomness, and then accepts if and only if all the calls to V_{VRS} accept. Hence, the algorithms Q_{aVRS} and D_{aVRS} for V_{aVRS} can easily be constructed using the algorithms Q_{VRS} and D_{VRS} for V_{VRS}.

Specifically, the algorithm Q_{aVRS} is defined as follows:

$$Q_{\text{aVRS}}\Big((I_\ell, \mathcal{B}_S, \mathcal{B}_H, d, \delta, s'), r_1 \cdots r_{\mathsf{rand}_{\text{aVRS}}(\ell,\kappa,\lambda,d,\delta,s')}, i\Big) \equiv$$

1. $m := \left\lceil \frac{\log(1-s')}{\log(1-s_{\text{VRS}}(\delta, 2^\kappa))} \right\rceil$.
2. Let $z \in \{1, \ldots, m\}$ be such that $i \in \big\{(z-1)\cdot\mathsf{query}_{\text{VRS}}(\ell,\kappa,\lambda,d)+1, \ldots, z\cdot\mathsf{query}_{\text{VRS}}(\ell,\kappa,\lambda,d)\big\}$.
3. $i_{\mathsf{new}} := i - z \cdot \mathsf{query}_{\text{VRS}}(\ell, \kappa, \lambda, d)$.
4. $r_1^{(i_{\mathsf{new}})} \cdots r_{\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)}^{(i_{\mathsf{new}})} := r_{(i_{\mathsf{new}}-1)\cdot\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)+1} \cdots r_{i_{\mathsf{new}}\cdot\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)}$.
5. $j := Q_{\text{VRS}}\big((I_\ell, \mathcal{B}_S, \mathcal{B}_H, d), r_1^{(i_{\mathsf{new}})} \cdots r_{\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)}^{(i_{\mathsf{new}})}, i_{\mathsf{new}}\big)$.
6. Output j.

The corresponding algorithm D_{aVRS} is defined as follows:

$$D_{\text{aVRS}}\Big((I_\ell, \mathcal{B}_S, d, \delta, s'), r_1 \cdots r_{\mathsf{rand}_{\text{aVRS}}(\ell,\kappa,\lambda,d,\delta,s')}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\text{aVRS}}(\ell,\kappa,\lambda,d,\delta,s')}\Big) \equiv$$

1. $m := \left\lceil \frac{\log(1-s')}{\log(1-s_{\text{VRS}}(\delta, 2^\kappa))} \right\rceil$.
2. Parse $\alpha_1 \cdots \alpha_{\mathsf{query}_{\text{aVRS}}(\ell,\kappa,\lambda,d,\delta,s')}$ as $\alpha_1^{(1)} \cdots \alpha_{\mathsf{query}_{\text{VRS}}(\ell,\kappa,\lambda,d)}^{(1)} \cdots \alpha_1^{(m)} \cdots \alpha_{\mathsf{query}_{\text{VRS}}(\ell,\kappa,\lambda,d)}^{(m)}$.
3. For $i = 1, \ldots, m$, do the following:
   (a) $r_1^{(i)} \cdots r_{\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)}^{(i)} := r_{(i-1)\cdot\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)+1} \cdots r_{i\cdot\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)}$;
   (b) $b_i := D_{\text{VRS}}\big((I_\ell, \mathcal{B}_S, \mathcal{B}_H, d), r_1^{(i)} \cdots r_{\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)}^{(i)}, \alpha_1^{(i)} \cdots \alpha_{\mathsf{query}_{\text{VRS}}(\ell,\kappa,\lambda,d)}^{(i)}\big)$.
4. $b := b_1 \wedge \cdots \wedge b_m$.
5. Output b.

The correctness of both Q_{aVRS} and D_{aVRS} easily follows by inspection of the algorithms and the above discussion, as well as Lemma 10.20. □

10.3.8 Non-Adaptivity of V_{ACSP}

Lemma 10.23. *V_{ACSP} is a non-adaptive PCP verifier.*

Proof. The PCP verifier V_{ACSP} is given as input an instance $(x, 1^t)$, allegedly in SuccinctACSP, and oracle access to a PCP proof π. The PCP proof π is parsed as a tuple $(p_0, \pi_0, p_1, \pi_1, \pi_{\mathsf{c}})$ where π_0 is a proof of proximity for p_0 to $\text{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathsf{m_H}(t)} - 1)$, π_1 is a proof of proximity for p_1 to $\text{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d_0^t + (2^{\mathsf{m_H}(t)} - 1)\sum_{i=1}^{\mathsf{c_N}(t)} d_i^t)$, and π_{c} is a proof of proximity for $p_0 - p_x$ to

$\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, S_m, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$ for a function p_x and subspace S_m depending on x. Finally, see Lemma B.8 for $\mathsf{query}_{\mathrm{ACSP}}(x,t)$, $\mathsf{rand}_{\mathrm{ACSP}}(x,t)$, and $\mathsf{length}_{\mathrm{ACSP}}(x,t)$.

The algorithm V_{ACSP} consists of five main steps: in the first step, V_{ACSP} generates the necessary "objects" needed for running subsequent steps; in the second step, V_{ACSP} tests proximity to RS (using a specific proximity parameter δ_{RS} and target soundness $s'_{\mathrm{RS}} = 0.5$); in the third step, V_{ACSP} tests proximity to VRS (using another specific proximity parameter δ_{VRS} and target soundness $s'_{\mathrm{VRS}} = 0.5$); in the fourth step, V_{ACSP} runs a consistency test between p_0 and p_1; in the fifth step, V_{ACSP} runs a consistency test of the instance $(x, 1^t)$ with p_0.

The algorithm Q_{ACSP} is then easily defined as follows:

$Q_{\mathrm{ACSP}}\Big((x, 1^t), r_1 \cdots r_{\mathsf{rand}_{\mathrm{ACSP}}(x,t)}, i\Big) \equiv$

1. Parameter instantiation:
   (a) $I_t := \textsc{FindIrrPoly}(1^{f(t)})$;
   (b) $\mathcal{B}_{\mathbb{F}_t} := \textsc{FieldBasis}(I_t)$;
   (c) $\mathcal{B}_{H_t} := \textsc{Find}\mathbf{H}(1^t)$;
   (d) for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $(a_{t,i}, b_{t,i}) := \textsc{Find}\mathbf{N}(1^t, i)$;
   (e) for $i = 0, 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $d_i^t := \textsc{Comp}\mathbf{D}(1^t, i)$;
   (f) $\big(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_t^{\mathbf{S}}(\mathsf{x})\big) := \textsc{Find}\mathbf{S}(1^t)$;
   (g) define $m := \lceil \log |x| \rceil$ and $\mathcal{B}_{S_m} := \big(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_m^{\mathbf{S}}(\mathsf{x})\big)$.
2. Deduce the amount of randomness and number of queries for the amplified verifiers:
   (a) $\delta_{\mathrm{RS}} := \frac{1}{8\mathsf{c}_{\mathbf{N}}(t)}$ and $s'_{\mathrm{RS}} := 0.5$;
   (b) $\mathsf{nq}_{\mathrm{aRS}} := \mathsf{query}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}})$;
   (c) $\mathsf{nr}_{\mathrm{aRS}} := \mathsf{rand}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}})$;
   (d) $\delta_{\mathrm{VRS}} := \frac{1}{8}$ and $s'_{\mathrm{VRS}} := 0.5$;
   (e) $\mathsf{nq}_{\mathrm{aVRS}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$;
   (f) $\mathsf{nr}_{\mathrm{aVRS}} := \mathsf{rand}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$;
   (g) $\delta_{\mathsf{c}} := \frac{1}{8\mathsf{c}_{\mathbf{N}}(t)}$ and $s'_{\mathsf{c}} := 0.5$;
   (h) $\mathsf{nq}_{\mathsf{c}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), m, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$;
   (i) $\mathsf{nr}_{\mathsf{c}} := \mathsf{rand}_{\mathrm{aVRS}}(f(t), f(t), m, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$.
3. If $i \in \big\{1, \ldots, \mathsf{nq}_{\mathrm{aRS}}\big\}$, then:
   (a) $r'_1 \cdots r'_{\mathsf{nr}_{\mathrm{aRS}}} := r_1 \cdots r_{\mathsf{nr}_{\mathrm{aRS}}}$;
   (b) $j_{\mathsf{old}} := Q_{\mathrm{aRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}}), r'_1 \cdots r'_{\mathsf{nr}_{\mathrm{aRS}}}, i\big)$;
   (c) $j := j_{\mathsf{old}}$;
   (d) output j.
4. If $i_{\mathsf{new}} \in \{1, \ldots, \mathsf{nq}_{\mathrm{aVRS}}\}$, where $i_{\mathsf{new}} := i - \mathsf{nq}_{\mathrm{aRS}}$, then:
   (a) $r''_1 \cdots r''_{\mathsf{nr}_{\mathrm{aVRS}}} := r_1 \cdots r_{\mathsf{nr}_{\mathrm{aVRS}}}$;
   (b) $j_{\mathsf{old}} := Q_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{H_t}, d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}}), r''_1 \cdots r''_{\mathsf{nr}_{\mathrm{aVRS}}}, i_{\mathsf{new}}\big)$;
   (c) $j := 2^{f(t)} + \mathsf{length}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}}) + j_{\mathsf{old}}$;
   (d) output j.
5. If $i_{\mathsf{new}} \in \{1, \ldots, \mathsf{c}_{\mathbf{N}}(t) + 1\}$, where $i_{\mathsf{new}} := i - \mathsf{nq}_{\mathrm{aRS}} - \mathsf{nq}_{\mathrm{aVRS}}$, then:
   (a) $r_1^{(\alpha)} \cdots r_{f(t)}^{(\alpha)} := r_1 \cdots r_{f(t)}$;
   (b) $\alpha := \textsc{GetRandElt}(\mathcal{B}_{\mathbb{F}_t}; r_1^{(\alpha)} \cdots r_{f(t)}^{(\alpha)})$;
   (c) if $i_{\mathsf{new}} \in \big\{1, \ldots, \mathsf{c}_{\mathbf{N}}(t)\big\}$, then:
      i. $(a_{t,i_{\mathsf{new}}}, b_{t,i_{\mathsf{new}}}) := \textsc{Find}\mathbf{N}(1^t, i_{\mathsf{new}})$;

ii. $\alpha_{i_{\mathsf{new}}} := \mathsf{aff}_{t,i_{\mathsf{new}}}(\alpha) = a_{t,i_{\mathsf{new}}} \cdot \alpha + b_{t,i_{\mathsf{new}}}$;
iii. $j_{\alpha_{i_{\mathsf{new}}}} := \textsc{GetIndexOfElt}(I_t, \mathcal{B}_{\mathbb{F}_t}, \alpha_{i_{\mathsf{new}}})$;
iv. $j := j_{\alpha_{i_{\mathsf{new}}}}$;
v. output j;

(d) if $i_{\mathsf{new}} = \mathsf{c}_{\mathbf{N}}(t) + 1$, then:
i. $j_\alpha := \textsc{GetIndexOfElt}(I_t, \mathcal{B}_{\mathbb{F}_t}, \alpha)$;
ii. $j := 2^{f(t)} + \mathsf{length}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}}) + j_\alpha$;
iii. output j.

6. If $i_{\mathsf{new}} \in \{1, \ldots, \mathsf{nq}_{\mathsf{c}}\}$, where $i_{\mathsf{new}} := i - \mathsf{nq}_{\mathrm{aRS}} - \mathsf{nq}_{\mathrm{aVRS}} - (\mathsf{c}_{\mathbf{N}}(t) + 1)$, then:
(a) $r''_1 \cdots r''_{\mathsf{nr}_{\mathsf{c}}} := r_1 \cdots r_{\mathsf{nr}_{\mathsf{c}}}$;
(b) $j_{\mathsf{old}} := Q_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{S_m}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}}), r''_1 \cdots r''_{\mathsf{nr}_{\mathsf{c}}}, i_{\mathsf{new}}\big)$;
(c) if $j \in \{1, \ldots, 2^{f(t)}\}$, then $j := j_{\mathsf{old}}$;
(d) if $j - 2^{f(t)} \in \{1, \ldots, \mathsf{length}_{\mathrm{aVRS}}(f(t), f(t), m, d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\mathsf{c}}, s'_{\mathsf{c}})\}$, then $j := \mathsf{length}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}})$
$+2^{f(t)}+\mathsf{length}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})+ j_{\mathsf{old}}$;
(e) output j.

The corresponding algorithm D_{ACSP} is defined as follows:

$D_{\mathrm{ACSP}}\Big((x, 1^t), r_1 \cdots r_{\mathsf{rand}_{\mathrm{ACSP}}(x,t)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{ACSP}}(x,t)}\Big) \equiv$

1. Parameter instantiation:
(a) $I_t := \textsc{FindIrrPoly}(1^{f(t)})$;
(b) $\mathcal{B}_{\mathbb{F}_t} := \textsc{FieldBasis}(I_t)$;
(c) $\mathcal{B}_{H_t} := \textsc{Find}\mathbf{H}(1^t)$;
(d) for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $(a_{t,i}, b_{t,i}) := \textsc{Find}\mathbf{N}(1^t, i)$;
(e) for $i = 0, 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $d_i^t := \textsc{Comp}\mathbf{D}(1^t, i)$;
(f) $\big(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_t^{\mathbf{S}}(\mathsf{x})\big) := \textsc{Find}\mathbf{S}(1^t)$;
(g) define $m := \lceil \log |x| \rceil$ and $\mathcal{B}_{S_m} := \big(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_m^{\mathbf{S}}(\mathsf{x})\big)$.
2. Deduce the amount of randomness and number of queries for the amplified verifiers:
(a) $\delta_{\mathrm{RS}} := \frac{1}{8\mathsf{c}_{\mathbf{N}}(t)}$ and $s'_{\mathrm{RS}} := 0.5$;
(b) $\mathsf{nq}_{\mathrm{aRS}} := \mathsf{query}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}})$;
(c) $\mathsf{nr}_{\mathrm{aRS}} := \mathsf{rand}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}})$;
(d) $\delta_{\mathrm{VRS}} := \frac{1}{8}$ and $s'_{\mathrm{VRS}} := 0.5$;
(e) $\mathsf{nq}_{\mathrm{aVRS}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$;
(f) $\mathsf{nr}_{\mathrm{aVRS}} := \mathsf{rand}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$;
(g) $\delta_{\mathsf{c}} := \frac{1}{8\mathsf{c}_{\mathbf{N}}(t)}$ and $s'_{\mathsf{c}} := 0.5$;
(h) $\mathsf{nq}_{\mathsf{c}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), m, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$;
(i) $\mathsf{nr}_{\mathsf{c}} := \mathsf{rand}_{\mathrm{aVRS}}(f(t), f(t), m, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$.
3. Parse $\alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{ACSP}}(x,t)}$ as

$$\alpha'_1 \cdots \alpha'_{\mathsf{nq}_{\mathrm{aRS}}} \alpha''_1 \cdots \alpha''_{\mathsf{nq}_{\mathrm{aVRS}}} \gamma_1 \cdots \gamma_{\mathsf{c}_{\mathbf{N}}(t)} \omega \tau_1 \cdots \tau_{\mathsf{nq}_{\mathsf{c}}} \ .$$

4. $r'_1 \cdots r'_{\mathsf{nr}_{\mathrm{aRS}}} := r_1 \cdots r_{\mathsf{nr}_{\mathrm{aRS}}}$.
5. $b_{\mathrm{RS}} := D_{\mathrm{aRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}}), r'_1 \cdots r'_{\mathsf{nr}_{\mathrm{aRS}}}, \alpha'_1 \cdots \alpha'_{\mathsf{nq}_{\mathrm{aRS}}}\big)$.
6. $r''_1 \cdots r''_{\mathsf{nr}_{\mathrm{aVRS}}} := r_1 \cdots r_{\mathsf{nr}_{\mathrm{aVRS}}}$.

7. $b_{\text{VRS}} := D_{\text{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{H_t}, d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\text{VRS}}, s'_{\text{VRS}}), r''_1 \cdots r''_{\mathsf{nr}_{\text{aVRS}}}, \alpha''_1 \cdots \alpha''_{\mathsf{nq}_{\text{aVRS}}}\big)$.
8. $r_1^{(\alpha)} \cdots r_{f(t)}^{(\alpha)} := r_1 \cdots r_{f(t)}$;
9. $\alpha := \text{GetRandElt}(\mathcal{B}_{\mathbb{F}_t}; r_1^{(\alpha)} \cdots r_{f(t)}^{(\alpha)})$.
10. $[P_t]^{\mathsf{A}} := \text{Find}\mathbf{P}(1^t)$.
11. $\omega' := [P_t]^{\mathsf{A}}(\alpha, \gamma_1, \ldots, \gamma_{\mathsf{c}_{\mathbf{N}}(t)})$.
12. $b_{\mathbf{P}} := (\omega' \stackrel{?}{=} \omega)$.
13. $r'''_1 \cdots r'''_{\mathsf{nr}_{\mathsf{c}}} := r_1 \cdots r_{\mathsf{nr}_{\mathsf{c}}}$.
14. $b_{\mathsf{c}} := D_{\text{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{S_m}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}}), r'''_1 \cdots r'''_{\mathsf{nr}_{\mathsf{c}}}, \tau_1 \cdots \tau_{\mathsf{nq}_{\mathsf{c}}}\big)$.
15. $b := b_{\text{RS}} \wedge b_{\text{VRS}} \wedge b_{\mathbf{P}} \wedge b_{\mathsf{c}}$.
16. Output b.

The correctness of both Q_{ACSP} and D_{ACSP} easily follows by inspection of the algorithms and the above discussion, as well as Lemma 10.21 and Lemma 10.22. □

10.4 Efficient Reverse Sampling

The third property considered by Barak and Goldreich [BG02, Definiton 3.2, third item] for a PCP verifier is the ability to efficiently find a random string that is consistent with a given query:

Definition 10.24 (Efficient Reverse Sampling)**.** *A PCP (resp., PCPP) verifier V is* **efficiently reverse samplable** *if V is a non-adaptive PCP (resp., PCPP) verifier, and hence can be decomposed into the pair of algorithms Q and D, and, moreover, there exists a probabilistic polynomial-time algorithm S such that, given any string x and positive integers i and j, $S(x, i, j)$ outputs a uniformly distributed string r that satisfies $Q(x, r, i) = j$.*

We prove that the PCP verifier V_{ACSP} does satisfy the definition above — that this is so is not clear, especially in light of the recursive nature of $V_{\text{RS},=}$, used as part of the construction of V_{ACSP}.

Claim 10.25. *The PCP verifier V_{ACSP} is efficiently reverse samplable.*

Again, the construction of V_{ACSP} is quite complicated, and we will have to proceed one step at a time. As before, our proof will be "bottom up". (More specific references will be given in each of the proofs.) Also, as we did in Section 10.3, we fix throughout a specific choice of parameters $(\eta, \kappa_0, \gamma, \mu)$, which parametrize V_{ACSP}.

Throughout, the symbol $\circ$ will denote string contatenation.

10.4.1 Efficient Reverse Sampling of $V_{\text{RS},=}$

Lemma 10.26. *The (strong) PCPP verifier $V_{\text{RS},=}$ is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma B.1. We have already established in Lemma 10.16 that $V_{\text{RS},=}$ is a non-adaptive (strong) PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_L)$, a query number $i \in \{1, \ldots, \mathsf{query}_{\text{RS},=}(\ell, \kappa)\}$, and a query index $j \in \{1, \ldots, |L| + \mathsf{length}_{\text{RS},=}(\ell, \kappa)\}$,

$$S_{\text{RS},=}\Big((I_\ell, \mathcal{B}_L), i, j\Big) \equiv$$

1. Compute $(r, n, t) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_L), i, j\big)$.
2. Output r.

Similarly to $Q_{\mathrm{RS},=}$, we have exported the "hard work" of $S_{\mathrm{RS},=}$ to an iterative procedure, RandSamp passing more state; on input an irreducible polynomial I_ℓ of degree ℓ with root x, a basis $\mathcal{B}_L$ for a κ-dimensional linear subset $L \subseteq \mathbb{F}_{2^\ell}$, a query number $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)\}$, and a query index $j \in \{1, \ldots, |L| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa)\}$, the procedure RandSamp outputs a triple (r, n, t), where $r \in \{0,1\}^{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$ is a uniformly distributed string satisfying $j = Q_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_L), r, i\big)$, n is the cardinality of all such strings r, and $t = \mathsf{rand}_{\mathrm{RS},=}(\ell, \kappa)$.

Essentially, deducing the algorithm RandSamp amounts to carefully examining the algorithm Translate, which was the iterative procedure doing the "hard work" of $Q_{\mathrm{RS},=}$, on known inputs I_ℓ, $\mathcal{B}_L$, and i but without knowing the input randomness r; of course (and this is the point), we do know that the output is $(\mathtt{text}, \mathsf{elt}, j)$, for some $\mathtt{text}$ and elt, and the goal of RandSamp is to find a string r that is consistent with the known inputs and output and, moreover, is in fact uniformly distributed among all such strings; it will be convenient to also keep track of the numbers n and t described in the previous paragraph.

At high level, the algorithm RandSamp will act differently, depending on whether the query j under consideration is to the implicit input $p\colon L \to \mathbb{F}_{2^\ell}$ or to the proximity proof $\pi = (f, \Pi)$; in the former case, j must be between 1 and $|L| = 2^\kappa$, while, in the latter case, $j - |L|$ must be between 1 and $|L_\beta| \cdot |L_1| + |L_1| \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \lfloor k/2 \rfloor - \gamma + \mu + 1) + |L'_0| \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \lceil k/2 \rceil + \gamma)$. In fact, if the query j is to the proximity proof, RandSamp will also distinguish between the case that j is a query to f, a query to one of the row-test proofs of proximity, and a query to one of the column-test proofs of proximity. We now describe the algorithm RandSamp in detail:

$\textsc{RandSamp}\big((I_\ell, \mathcal{B}_L), i, j\big) \equiv$

1. if $\kappa \le \kappa_0$, then output $(\varepsilon, 0, 0)$.

   indeed, in the base case of $Q_{\mathrm{RS},=}$, no randomness is used, and thus here none need be sampled
2. if $\kappa > \kappa_0$, then do the following:
   (a) $m_0 := \lceil \kappa/2 \rceil + \gamma$, $m_1 := \lfloor \kappa/2 \rfloor - \gamma + \mu$;
   (b) $m := 1 + \max\{m_0, m_1\}$;
   (c) $\mathcal{B}_{L_0} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma})$;
   (d) $\mathcal{B}_{L'_0} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu})$;
   (e) $\mathcal{B}_{L_1} := (a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_\kappa)$;
   (f) $[Z_{L_0}]^{\mathsf{A}} := \textsc{FindSubspPoly}(I_\ell, \mathcal{B}_{L_0})$;
   (g) $\mathcal{B}_{Z_{L_0}(L_1)} := ([Z_{L_0}]^{\mathsf{A}}(a_{\lfloor \kappa/2 \rfloor - \gamma + 1}), \ldots, [Z_{L_0}]^{\mathsf{A}}(a_\kappa))$;
   (h) if $j' \in \{1, \ldots, 2^\kappa\}$, where $j' := j$, then do the following:
      i.e., j is a query into the implicit input $p\colon L \to \mathbb{F}_{2^\ell}$
      i. $\alpha := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_L, j')$;
      ii. deduce $\beta \in L_1$ such that $[Z_{L_0}]^{\mathsf{A}}(\beta) = [Z_{L_0}]^{\mathsf{A}}(\alpha)$;
      iii. if $\beta \in L'_0$, then $\mathcal{B}_{L_\beta} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1})$;
      iv. if $\beta \notin L'_0$, then $\mathcal{B}_{L_\beta} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}, \beta)$;
      v. if $\alpha \notin L'_0$, then do the following:
         A. $r_1 := 0$;
         B. $r_2 \cdots r_{1+m_0} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L_1}, \beta)$;
         C. $\widetilde{m}_0 := m - 1 - m_0$;
         D. if $\widetilde{m} > 0$, draw $r_{m-\widetilde{m}_0+1}, \ldots, r_m \in \{0,1\}$ at random;
         E. $j'' := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_{L_\beta}, \alpha)$;

F. $(\overline{r}_0, n_0, \ell_0) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L_\beta}), i, j''\big)$;

G. output $(r_1 \cdots r_m \circ \overline{r}_0, 2^{\widetilde{m}_0} \cdot n_0, 2^m \cdot \ell_0)$;

vi. if $\alpha \in L'_0$, then do the following:

first figure out the probability mass to assign to a row test

A. $r_2^{(0)} \cdots r_{1+m_0}^{(0)} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L_1}, \beta)$;

B. $\widetilde{m}_0 := m - 1 - m_0$;

C. if $\widetilde{m}_0 > 0$, draw $r_{m-\widetilde{m}_0+1}^{(0)}, \ldots, r_m^{(0)} \in \{0,1\}$ at random;

D. $j'' := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_{L_\beta}, \alpha)$;

E. $(\overline{r}_0, n_0, \ell_0) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L_\beta}), i, j''\big)$;

then figure out probability mass to assign to a column test

F. $r_2^{(1)} \cdots r_{1+m_1}^{(1)} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L'_0}, \alpha)$;

G. $\widetilde{m}_1 := m - 1 - m_1$;

H. if $\widetilde{m}_1 > 0$, draw $r_{m-\widetilde{m}_1+1}^{(1)}, \ldots, r_m^{(1)} \in \{0,1\}$ at random;

I. $j'' := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, \alpha)$;

J. $(\overline{r}_1, n_1, \ell_1) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L'_0}), i, j''\big)$;

finally put everything together

K. $r_1 \cdots r_m \circ \overline{r}$ is set to $r_1^{(0)} \cdots r_m^{(0)} \circ \overline{r}_0$ w.p. $\frac{2^{\widetilde{m}_0} \cdot n_0}{2^m \cdot \ell_0}$ and to $r_1^{(1)} \cdots r_m^{(1)} \circ \overline{r}_1$ w.p. $\frac{2^{\widetilde{m}_1} \cdot n_1}{2^m \cdot \ell_1}$;

L. output $(r_1 \cdots r_m \circ \overline{r}, 2^{\widetilde{m}_0} \cdot n_0 + 2^{\widetilde{m}_1} \cdot n_1, 2^m \cdot \ell_0 + 2^m \cdot \ell_1)$;

(i) if $j' \in \{1, \ldots, 2^{\kappa+\mu+1}\}$, where $j' := j - 2^\kappa$, then do the following:

i.e., j is a query into the bivariate evaluation f

i. $\iota_{\mathsf{row}} := \lfloor j' / 2^{\lceil \kappa/2 \rceil + \gamma} \rfloor$;

ii. $\iota_{\mathsf{col}} := j' - \iota_{\mathsf{col}} \cdot 2^{\lceil \kappa/2 \rceil + \gamma}$;

iii. $\alpha := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_{L'_0}, \iota_{\mathsf{col}})$;

iv. $\beta := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, \iota_{\mathsf{row}})$;

first figure out probability mass to assign to a row test

v. $r_2^{(0)} \cdots r_{1+m_0}^{(0)} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L_1}, \beta)$;

vi. $\widetilde{m}_0 := m - 1 - m_0$;

vii. if $\widetilde{m}_0 > 0$, draw $r_{m-\widetilde{m}_0+1}^{(0)}, \ldots, r_m^{(0)} \in \{0,1\}$ at random;

viii. $j'' := \iota_{\mathsf{col}}$;

ix. $(\overline{r}_0, n_0, \ell_0) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L_\beta}), i, j''\big)$;

then figure out probability mass to assign to a column test

x. $r_2^{(1)} \cdots r_{1+m_1}^{(1)} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L'_0}, \alpha)$;

xi. $\widetilde{m}_1 := m - 1 - m_1$;

xii. if $\widetilde{m}_1 > 0$, draw $r_{m-\widetilde{m}_1+1}^{(1)}, \ldots, r_m^{(1)} \in \{0,1\}$ at random;

xiii. $j'' := \iota_{\mathsf{row}}$;

xiv. $(\overline{r}_1, n_1, \ell_1) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L'_0}), i, j''\big)$;

finally put everything together

xv. $r_1 \cdots r_m \circ \overline{r}$ is set to $r_1^{(0)} \cdots r_m^{(0)} \circ \overline{r}_0$ w.p. $\frac{2^{\widetilde{m}_0} \cdot n_0}{2^m \cdot \ell_0}$ and to $r_1^{(1)} \cdots r_m^{(1)} \circ \overline{r}_1$ w.p. $\frac{2^{\widetilde{m}_1} \cdot n_1}{2^m \cdot \ell_1}$;

xvi. output $(r_1 \cdots r_m \circ \overline{r}, 2^{\widetilde{m}_0} \cdot n_0 + 2^{\widetilde{m}_1} \cdot n_1, 2^m \cdot \ell_0 + 2^m \cdot \ell_1)$;

(j) if $j' \in \{1, \ldots, 2^{\lceil \kappa/2 \rceil + \gamma}\}$, where $j' := \left\lfloor \frac{j - 2^\kappa - 2^{\kappa+\mu+1}}{\mathsf{length}_{\mathrm{RS},=}(\ell, \lfloor \kappa/2 \rfloor - \gamma + \mu + 1)} \right\rfloor$, then do the following:

i.e., j is a query into a row proximity sub-proof

i. $r_1 := 0$;

ii. $\beta' := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, j')$;

iii. deduce $\beta \in L_1$ such that $[Z_{L_0}]^{\mathsf{A}}(\beta) = \beta'$;
iv. $r_2 \cdots r_{1+m_0} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L_1}, \beta)$;
v. $\widetilde{m}_0 := m - 1 - m_0$;
vi. if $\widetilde{m}_0 > 0$, draw $r_{m-\widetilde{m}_0+1}, \ldots, r_m \in \{0,1\}$ at random;
vii. $j'' := 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1} + (j - 2^\kappa - 2^{\kappa+\mu+1} - j' \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \lfloor \kappa/2 \rfloor - \gamma + \mu + 1))$;
viii. $(\overline{r}_0, n_0, \ell_0) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L_\beta}), i, j''\big)$;
ix. output $(r_1 \cdots r_m \circ \overline{r}_0, 2^{\widetilde{m}_0} \cdot n_0, 2^m \cdot \ell_0)$;

(k) if $j' \in \{1, \ldots, 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu}\}$, where $j' := \left\lfloor \frac{j - 2^\kappa - 2^{\kappa+\mu+1} - 2^{\lceil \kappa/2 \rceil + \gamma} \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \lfloor \kappa/2 \rfloor - \gamma + \mu + 1)}{\mathsf{length}_{\mathrm{RS},=}(\ell, \lceil \kappa/2 \rceil + \mu)} \right\rfloor$, then do the following:

i.e., j is a query into a column proximity sub-proof

i. $r_1 := 1$;
ii. $\alpha := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_{L_0'}, j')$;
iii. deduce $\beta \in L_1$ such that $[Z_{L_0}]^{\mathsf{A}}(\beta) = [Z_{L_0}]^{\mathsf{A}}(\alpha)$;
iv. $r_2 \cdots r_{1+m_1} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L_0'}, \alpha)$;
v. $\widetilde{m}_1 := m - 1 - m_1$;
vi. if $\widetilde{m}_1 > 0$, draw $r_{m-\widetilde{m}_1+1}, \ldots, r_m \in \{0,1\}$ at random;
vii. $j'' := 2^{\lceil \kappa/2 \rceil + \mu} - (j - 2^\kappa - 2^{\kappa+\mu+1} - 2^{\lceil \kappa/2 \rceil + \gamma} \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \lfloor \kappa/2 \rfloor - \gamma + \mu + 1) - j' \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \lceil \kappa/2 \rceil + \gamma))$;
viii. $(\overline{r}_1, n_1, \ell_1) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}), i, j''\big)$;
ix. output $(r_1 \cdots r_m \circ \overline{r}_1, 2^{\widetilde{m}_1} \cdot n_1, 2^m \cdot \ell_1)$.

Note that RandSamp happens to be independent of i. Also recall (see Lemma B.10) that

$$\begin{aligned}\mathsf{length}_{\mathrm{RS},=}(\ell, \kappa) &= 2^{\lceil \kappa/2 \rceil + \gamma} \cdot 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1} \\ &\quad + 2^{\lceil \kappa/2 \rceil + \gamma} \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \lfloor \kappa/2 \rfloor - \gamma + \mu + 1) + 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu} \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \lceil \kappa/2 \rceil + \gamma) \ .\end{aligned}$$

The correctness of $S_{\mathrm{RS},=}$ follows from the correctness of RandSamp, which follows by inspection. □

We remark that in Step 2(h)ii, Step 2(j)iii, and Step 2(k)iii, RandSamp is required to "invert" the polynomial Z_{L_0}. More precisely, given $\beta' \in Z_{L_0}(\mathbb{F}_{2^\ell})$, RandSamp needs to compute some $\beta \in L_1$ such that $Z_{L_0}(\beta) = \beta'$. Even though Z_{L_0} is a high-degree polynomial, the same fact that implies the existence of a small arithmetic circuit $[Z_{L_0}]^{\mathsf{A}}$ for computing Z_{L_0} also implies a "sparse" computation for finding β. Specifically, because Z_{L_0} is a linearized polynomial, the map $Z_{L_0} \colon \mathbb{F}_{2^\ell} \to \mathbb{F}_{2^\ell}$ is a linear map. Hence, finding β reduces to finding the (unique) solution in L_1 of a $\dim(L_0)$-dimensional linear system $Mx = \beta'$ where M is the matrix induced by the linear map Z_{L_0}. This can be done in time that is polynomial in ℓ and κ, and thus β can indeed be found efficiently by RandSamp.

10.4.2 Efficient Reverse Sampling of $V_{\mathrm{RS},<}$

Lemma 10.27. *The (strong) PCPP verifier $V_{\mathrm{RS},<}$ is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma B.2. We have already established in Lemma 10.17 that $V_{\mathrm{RS},<}$ is a non-adaptive (strong) PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, d)$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},<}(\ell, \kappa, d)\}$, and $j \in \{1, \ldots, |S| + \mathsf{length}_{\mathrm{RS},<}(\ell, \kappa, d)\}$,

$S_{\mathrm{RS},<}\Big((I_\ell, \mathcal{B}_S, d), i, j\Big) \equiv$

1. If $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)\}$, then:
   (a) compute $r := S_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S), i, j\big)$.
2. If $i' \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)\}$, where $i' := i - \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)$, then:
   (a) if $j \in \{1, \ldots, |S|\}$, then:
      i. compute $r := S_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S), i', j\big)$;
   (b) if $j' \in \{|S| + 1, \ldots, |S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa)\}$, where $j' := j - \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa)$, then:
      i. compute $r := S_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S), i', j'\big)$.
3. Output r.

Recall that $\mathsf{rand}_{\mathrm{RS},<}(\ell, \kappa, d) = \mathsf{rand}_{\mathrm{RS},=}(\ell, \kappa)$, and thus r is always of the correct length (and thus there is never a need to pad it with random bits). The correctness of $S_{\mathrm{RS},<}$ easily follows by inspection, as well as from Lemma 10.16. □

10.4.3 Efficient Reverse Sampling of $V_{\mathrm{RS},>}$

Lemma 10.28. *The (strong) PCPP verifier $V_{\mathrm{RS},>}$ is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma B.3. We have already established in Lemma 10.18 that $V_{\mathrm{RS},>}$ is a non-adaptive (strong) PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, d)$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},>}(\ell, \kappa, d)\}$, and $j \in \{1, \ldots, |S| + \mathsf{length}_{\mathrm{RS},>}(\ell, \kappa, d)\}$,

$S_{\mathrm{RS},>}\Big((I_\ell, \mathcal{B}_S, d), i, j\Big) \equiv$

1. If $d \geq 2^\kappa$, then output $\perp$. (Because the decision algorithm $D_{\mathrm{RS},>}$ will accept without examining any query answer, and thus the query algorithm $Q_{\mathrm{RS},>}$ will in this case not make any queries.)
2. If $d < 2^\kappa$, then:
   (a) $d_{\kappa,\eta} := |S|/2^\eta - 1$;
   (b) $m_{\kappa,\eta,d} := \lfloor (d+1)/(d_{\kappa,\eta} + 1) \rfloor$;
   (c) $d_{\kappa,\eta,d} := d - m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta} + 1)$;
   (d) if $i \in \big\{1, \ldots, m_{\kappa,\eta,d} \cdot \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)\big\}$, then:
      i. $z := \lfloor (i-1)/\mathsf{query}_{\mathrm{RS},=}(\ell, \kappa) \rfloor$;
      ii. $i_{\mathsf{new}} := i - z \cdot \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)$;
      iii. $j_{\mathsf{new}} := j - \big(|S| + z \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa))\big)$;
      iv. $r := S_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$;
      v. output r;
   (e) if $m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta} + 1) < d + 1$ and $i_{\mathsf{new}} \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},<}(\ell, \kappa, d_{\kappa,\eta,d})\}$, where $i_{\mathsf{new}} := i - m_{\kappa,\eta,d} \cdot \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)$, then:
      i. $j_{\mathsf{new}} := j - \big(|S| + m_{\kappa,\eta,d} \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa))\big)$;
      ii. $r := S_{\mathrm{RS},<}\big((I_\ell, \mathcal{B}_S, d_{\kappa,\eta,d}), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$;
      iii. output r;
   (f) if $m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta} + 1) = d + 1$ and $i_{\mathsf{new}} \in \{1, \ldots, (1 + m_{\kappa,\eta,d})\}$, where $i_{\mathsf{new}} := i - m_{\kappa,\eta,d} \cdot \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)$, then:
      i. if $i_{\mathsf{new}} = 1$, then:
         (it must be that $j \in \{1, \ldots, |S|\}$)
         A. $j_{\mathsf{new}} := j$;

B. $\gamma := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, j_{\mathsf{new}})$;
C. $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \gamma)$;
D. output r;

ii. if $i_{\mathsf{new}} \in \{2, \ldots, (1 + m_{\kappa,\eta,d})\}$, then:
(it must be that $j \in \{|S| + (i_{\mathsf{new}} - 2) \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa)) + 1, \ldots, |S| + (i_{\mathsf{new}} - 2) \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa)) + |S|\}$)
A. $j_{\mathsf{new}} := j - \big(|S| + (i_{\mathsf{new}} - 2) \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa))\big)$;
B. $\tau := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, j_{\mathsf{new}})$;
C. $\gamma := \tau^{1/((i_{\mathsf{new}}-2)\cdot(d_{\kappa,\eta}+1))}$;
D. $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \gamma)$;
E. output r;

(g) if $m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta} + 1) < d + 1$ and $i_{\mathsf{new}} \in \{1, \ldots, (1 + m_{\kappa,\eta,d})\}$, where $i_{\mathsf{new}} := i - (m_{\kappa,\eta,d} \cdot \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa) + \mathsf{query}_{\mathrm{RS},<}(\ell, \kappa, d_{\kappa,\eta,d}))$, then:

i. if $i_{\mathsf{new}} = 1$, then:
(it must be that $j \in \{1, \ldots, |S|\}$)
A. $j_{\mathsf{new}} := j$;
B. $\gamma := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, j_{\mathsf{new}})$;
C. $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \gamma)$;
D. output r;

ii. if $i_{\mathsf{new}} \in \{2, \ldots, (1 + m_{\kappa,\eta,d} + 1)\}$, then:
(it must be that $j \in \{|S| + (i_{\mathsf{new}} - 2) \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa)) + 1, \ldots, |S| + (i_{\mathsf{new}} - 2) \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa)) + |S|\}$)
A. $j_{\mathsf{new}} := j - \big(|S| + (i_{\mathsf{new}} - 2) \cdot (|S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa))\big)$;
B. $\tau := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, j_{\mathsf{new}})$;
C. $\gamma := \tau^{1/((i_{\mathsf{new}}-2)\cdot(d_{\kappa,\eta}+1))}$;
D. $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \gamma)$;
E. output r.

Recall that $\mathsf{rand}_{\mathrm{RS},>}(\ell, \kappa, d) = \max\{\mathsf{rand}_{\mathrm{RS},=}(\ell, \kappa), \mathbf{1}_{\kappa,\eta,d} \cdot \mathsf{rand}_{\mathrm{RS},<}(\ell, \kappa, d_{\kappa,\eta,d})\}$, where $\mathbf{1}_{\kappa,\eta,d} := 1$ if $(d+1) > d_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1)$ and $\mathbf{1}_{\kappa,\eta,d} := 0$ otherwise, and thus r is always of the correct length (and thus there is never a need to pad it with random bits). The correctness of $S_{\mathrm{RS},>}$ easily follows by inspection, as well as from Lemma 10.16 and Lemma 10.17. □

10.4.4 Efficient Reverse Sampling of V_{RS}

Lemma 10.29. *The (strong) PCPP verifier V_{RS} is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma B.4. We have already established in Lemma 10.19 that V_{RS} is a non-adaptive (strong) PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, d)$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d)\}$, and $j \in \{1, \ldots, |S| + \mathsf{length}_{\mathrm{RS}}(\ell, \kappa, d)\}$,

$S_{\mathrm{RS}}\Big((I_\ell, \mathcal{B}_S, d), i, j\Big) \equiv$

1. If $\kappa \leq \eta$:
   (a) Output a random string in $\mathsf{rand}_{\mathrm{RS}}(\ell, \kappa, d)$.
2. If $\kappa > \eta$:
   (a) Set $d_{\kappa,\eta} := |S|/2^\eta - 1$.

(b) If $d < d_{\kappa,\eta}$, then output $r := S_{\mathrm{RS},<}\big((I_\ell, \mathcal{B}_S, d), i, j\big)$.
(c) If $d = d_{\kappa,\eta}$, then output $r := S_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S), i, j\big)$.
(d) If $d > d_{\kappa,\eta}$, then output $r := S_{\mathrm{RS},>}\big((I_\ell, \mathcal{B}_S, d), i, j\big)$.
(e) Pad r with random bits until it is $\mathsf{rand}_{\mathrm{RS}}(\ell, \kappa)$-bits long.
(f) Output r.

The correctness of S_{RS} easily follows by inspection, as well as from Lemma 10.16, Lemma 10.17, and Lemma 10.18. □

10.4.5 Efficient Reverse Sampling of V_{VRS}

Lemma 10.30. *The (strong) PCPP verifier V_{VRS} is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma B.5. We have already established in Lemma 10.20 that V_{VRS} is a non-adaptive (strong) PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, \mathcal{B}_H, d)$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{VRS}}(\ell, \kappa, \lambda, d)\}$, and $j \in \{1, \ldots, |S| + \mathsf{length}_{\mathrm{VRS}}(\ell, \kappa, \lambda, d)\}$,

$$S_{\mathrm{VRS}}\Big((I_\ell, \mathcal{B}_S, \mathcal{B}_H, d), i, j\Big) \equiv$$

1. If $i \in \big\{1, \ldots, \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d - |H|)\big\}$, then:
   (it must be that $j \in \{|S|+1, \ldots, |S| + |S| + \mathsf{length}_{\mathrm{RS}}(\ell, \kappa, d - |H|)\}$)
   (a) $i_{\mathsf{new}} := i$;
   (b) $j_{\mathsf{new}} := j - |S|$;
   (c) $r := S_{\mathrm{RS}}\big((I_\ell, \mathcal{B}_S, d - |H|), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$.
2. If $i = \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d - |H|) + 1$, then:
   (it must be that $j \in \{1, \ldots, |S|\}$)
   (a) $j_{\mathsf{new}} := j$;
   (b) $\alpha := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, j_{\mathsf{new}})$;
   (c) $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \alpha)$.
3. If $i = \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d - |H|) + 2$, then:
   (it must be that $j \in \{|S|+1, \ldots, |S| + |S|\}$)
   (a) $j_{\mathsf{new}} := j - |S|$;
   (b) $\alpha := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, j_{\mathsf{new}})$;
   (c) $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \alpha)$.
4. Pad r with random bits until it is $\mathsf{rand}_{\mathrm{VRS}}(\ell, \kappa, \lambda, d)$-bits long.
5. Output r.

The correctness of S_{VRS} easily follows by inspection, as well as from Lemma 10.19. □

10.4.6 Efficient Reverse Sampling of V_{aRS}

Lemma 10.31. *The PCPP verifier V_{aRS} is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma B.6. We have already established in Lemma 10.21 that V_{aRS} is a non-adaptive PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, d, \delta, s')$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{aRS}}(\ell, \kappa, d, \delta, s')\}$, and $j \in \{1, \ldots, |S| + \mathsf{length}_{\mathrm{aRS}}(\ell, \kappa, d, \delta, s')\}$,

$$S_{\mathrm{aRS}}\Big((I_\ell, \mathcal{B}_S, d, \delta, s'), i, j\Big) \equiv$$

1. $m := \left\lceil \frac{\log(1-s')}{\log(1-s_{\mathrm{RS}}(\delta,2^\kappa))} \right\rceil$.
2. Let $z \in \{1,\ldots,m\}$ be such that $i \in \big\{(z-1)\cdot \mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)+1,\ldots,z\cdot \mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)\big\}$.
3. $i_{\mathsf{new}} := i-(z-1)\cdot \mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)$.
4. $j_{\mathsf{new}} := j$.
5. $r_z := S_{\mathrm{RS}}\big((I_\ell,\mathcal{B}_S,d),i_{\mathsf{new}},j_{\mathsf{new}}\big)$.
6. For $l \in \{1,\ldots,m\}-\{z\}$, draw a random $\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)$-bit string r_l.
7. $r := r_1 \cdots r_m$.
8. Output r.

The correctness of S_{aRS} easily follows by inspection, as well as from Lemma 10.19. □

10.4.7 Efficient Reverse Sampling of V_{aVRS}

Lemma 10.32. *The PCPP verifier V_{aVRS} is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma B.7. We have already established in Lemma 10.22 that V_{aVRS} is a non-adaptive PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell,\mathcal{B}_S,\mathcal{B}_H,d,\delta,s')$, $i \in \{1,\ldots,\mathsf{query}_{\mathrm{aVRS}}(\ell,\kappa,\lambda,d,\delta,s')\}$, and $j \in \{1,\ldots,|S|+\mathsf{length}_{\mathrm{aVRS}}(\ell,\kappa,\lambda,d,\delta,s')\}$,

$S_{\mathrm{aVRS}}\Big((I_\ell,\mathcal{B}_S,\mathcal{B}_H,d,s'),i,j\Big) \equiv$

1. $m := \left\lceil \frac{\log(1-s')}{\log(1-s_{\mathrm{VRS}}(\delta,2^\kappa))} \right\rceil$.
2. Let $z \in \{1,\ldots,m\}$ be such that $i \in \big\{(z-1)\cdot \mathsf{query}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)+1,\ldots,z\cdot \mathsf{query}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)\big\}$.
3. $i_{\mathsf{new}} := i-(z-1)\cdot \mathsf{query}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)$.
4. $j_{\mathsf{new}} := j$.
5. $r_z := S_{\mathrm{VRS}}\big((I_\ell,\mathcal{B}_S,\mathcal{B}_H,d),i_{\mathsf{new}},j_{\mathsf{new}}\big)$.
6. For $l \in \{1,\ldots,m\}-\{z\}$, draw a random $\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)$-bit string r_l.
7. $r := r_1 \cdots r_m$.
8. Output r.

The correctness of S_{aVRS} easily follows by inspection, as well as from Lemma 10.20. □

10.4.8 Efficient Reverse Sampling of V_{ACSP}

Lemma 10.33. *The PCP verifier V_{ACSP} is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma B.8. We have already established in Lemma 10.23 that V_{aVRS} is a non-adaptive PCP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(x,1^t)$, $i \in \{1,\ldots,\mathsf{query}_{\mathrm{ACSP}}(x,t)\}$, and $j \in \{1,\ldots,\mathsf{length}_{\mathrm{ACSP}}(x,t)\}$,

$S_{\mathrm{ACSP}}\Big((x,1^t),i,j\Big) \equiv$

1. Parameter instantiation:
   (a) $I_t := \textsc{FindIrrPoly}(1^{f(t)})$;
   (b) $\mathcal{B}_{\mathbb{F}_t} := \textsc{FieldBasis}(I_t)$;
   (c) $\mathcal{B}_{H_t} := \textsc{Find}\mathbf{H}(1^t)$;
   (d) for $i=1,\ldots,\mathsf{c_N}(t)$, $(a_{t,i},b_{t,i}) := \textsc{Find}\mathbf{N}(1^t,i)$;
   (e) for $i=0,1,\ldots,\mathsf{c_N}(t)$, $d_i^t := \textsc{Comp}\mathbf{D}(1^t,i)$;
   (f) $\big(\alpha_1^{\mathsf{S}}(\mathsf{x}),\ldots,\alpha_t^{\mathsf{S}}(\mathsf{x})\big) := \textsc{Find}\mathbf{S}(1^t)$;

(g) define $m := \lceil \log |x| \rceil$ and $\mathcal{B}_{S_m} := \big(\alpha_1^{\mathbf{S}}(\mathsf{x}), \ldots, \alpha_m^{\mathbf{S}}(\mathsf{x})\big)$.

2. Deduce the number of queries for the amplified verifiers:
   (a) $\delta_{\mathrm{RS}} := \frac{1}{8\mathsf{c}_{\mathbf{N}}(t)}$ and $s'_{\mathrm{RS}} := 0.5$;
   (b) $\mathsf{nq}_{\mathrm{aRS}} := \mathsf{query}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}})$;
   (c) $\delta_{\mathrm{VRS}} := \frac{1}{8}$ and $s'_{\mathrm{VRS}} := 0.5$;
   (d) $\mathsf{nq}_{\mathrm{aVRS}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$;
   (e) $\delta_{\mathsf{c}} := \frac{1}{8\mathsf{c}_{\mathbf{N}}(t)}$ and $s'_{\mathsf{c}} := 0.5$;
   (f) $\mathsf{nq}_{\mathsf{c}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), m, d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$.
3. If $i \in \{1, \ldots, \mathsf{nq}_{\mathrm{aRS}}\}$, then:
   (a) $i_{\mathsf{new}} := i$;
   (b) $j_{\mathsf{new}} := j$;
   (c) $r := S_{\mathrm{aRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}}), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$.
4. If $i_{\mathsf{new}} \in \{1, \ldots, \mathsf{nq}_{\mathrm{aVRS}}\}$, where $i_{\mathsf{new}} := i - \mathsf{nq}_{\mathrm{aRS}}$, then:
   (a) $j_{\mathsf{new}} := j - 2^{\mathsf{m}_{\mathbf{H}}(t)} - \mathsf{length}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}})$;
   (b) $r := S_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{H_t}, d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}}), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$.
5. If $i_{\mathsf{new}} \in \{1, \ldots, \mathsf{c}_{\mathbf{N}}(t) + 1\}$, where $i_{\mathsf{new}} := i - \mathsf{nq}_{\mathrm{aRS}} - \mathsf{nq}_{\mathrm{aVRS}}$, then:
   (a) if $i_{\mathsf{new}} \in \{1, \ldots, \mathsf{c}_{\mathbf{N}}(t)\}$, then:
      i. $j_{\mathsf{new}} := j$;
      ii. $\alpha_{i_{\mathsf{new}}} := \textsc{GetEltWithIndex}(I_t, \mathcal{B}_{\mathbb{F}_t}, j_{\mathsf{new}})$;
      iii. $(a_{t,i_{\mathsf{new}}}, b_{t,i_{\mathsf{new}}}) := \textsc{FindN}(1^t, i_{\mathsf{new}})$;
      iv. $\alpha := \mathsf{aff}^{-1}_{t,i_{\mathsf{new}}}(\alpha_{i_{\mathsf{new}}}) = a^{-1}_{t,i_{\mathsf{new}}} \cdot (\alpha - b_{t,i_{\mathsf{new}}})$;
      v. $r := \textsc{GetRandFromElt}(I_t, \mathcal{B}_{\mathbb{F}_t}, \alpha)$;
   (b) if $i_{\mathsf{new}} = \mathsf{c}_{\mathbf{N}}(t) + 1$, then:
      i. $j_{\mathsf{new}} := j$;
      ii. $\alpha := \textsc{GetEltWithIndex}(I_t, \mathcal{B}_{\mathbb{F}_t}, j_{\mathsf{new}})$;
      iii. $r := \textsc{GetRandFromElt}(I_t, \mathcal{B}_{\mathbb{F}_t}, \alpha)$.
6. If $i_{\mathsf{new}} \in \{1, \ldots, \mathsf{nq}_{\mathsf{c}}\}$, where $i_{\mathsf{new}} := i - \mathsf{nq}_{\mathrm{aRS}} - \mathsf{nq}_{\mathrm{aVRS}} - (\mathsf{c}_{\mathbf{N}}(t) + 1)$, then:
   (a) if $j \in \{1, \ldots, 2^{f(t)}\}$, then:
      i. compute $r := S_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{S_m}, \delta_{\mathsf{c}}, s'_{\mathsf{c}}), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$;
   (b) if $j' \in \{2^{f(t)} + 1, \ldots, 2^{f(t)} + \mathsf{length}_{\mathrm{aVRS}}(f(t), f(t), m, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})\}$,
   where $j' := j - \mathsf{length}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}})$
   $-2^{f(t)} - \mathsf{length}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$,
   then:
      i. compute $r := S_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{S_m}, \delta_{\mathsf{c}}, s'_{\mathsf{c}}), i_{\mathsf{new}}, j'\big)$;
7. Pad r with random bits until it is $\mathsf{rand}_{\mathrm{ACSP}}(x, t)$-bits long.
8. Output r.

The correctness of S_{ACSP} easily follows by inspection, as well as from Lemma 10.21 and Lemma 10.22. □

10.5 Proof Of Knowledge

The fourth property considered by Barak and Goldreich [BG02, Definiton 3.2, third item] for a PCP verifier is a proof-of-knowledge property. We show that the PCP verifier V_{ACSP} satisfies a *variant* of the proof-of-knowledge property considered by Barak and Goldreich. At high-level, the original definition of Barak and Goldreich requires the following: if, on input some string x and with access to some oracle π, the PCP verifier accepts with high enough probability, then

π "contains" a witness for x, in the sense that there is an efficient *knowledge extractor* that, on input x and with oracle access to π, can compute any bit of the witness with high probability. Formally:

Definition 10.34 (PCP Proof of Knowledge)**.** *Fix $\varepsilon\colon \mathbb{N} \to [0,1]$. A PCP verifier V has the* **proof-of-knowledge property with error** *ε if there exists a probabilistic polynomial-time oracle machine E_{PCP} such that the following holds: for every two strings x and π, if $\Pr[V^{\pi}(x) = 1] > \varepsilon(|x|)$ then there exists $w = w_1 \cdots w_T$ such that $(x,w) \in R$ and, for $i = 1,\ldots,T$, $\Pr[E^{\pi}_{\mathrm{PCP}}(x,i) = w_i] > 2/3$.*

A weaker definition does not require the knowledge extractor E_{PCP} to be an *implicit* representation of the witness, and instead allows E_{PCP} to run in time that is polynomial in the witness length:

Definition 10.35 (Explicit PCP Proof of Knowledge)**.** *Fix $\varepsilon\colon \mathbb{N} \to [0,1]$. A PCP verifier V has the* **explicit proof-of-knowledge property with error** *ε if there exists a probabilistic polynomial-time oracle machine E_{PCP} such that the following holds: for every two strings x and π, if $\Pr[V^{\pi}(x) = 1] > \varepsilon(|x|)$ then there exists $w = w_1 \cdots w_T$ such that $(x,w) \in R$ and $\Pr[E^{\pi}_{\mathrm{PCP}}(x,1^T) = w] > 2/3$.*

We prove that V_{ACSP} satisfies this weaker definition (and then in Remark 10.37 we explain why it does *not* satisfy the original one). Later, in Section 10.6, we will show that the weak PCP proof-of-knowledge property still suffices for constructing a variant of universal arguments (with a correspondingly weaker proof of knowledge property), which are still useful for a number of applications.

Claim 10.36. *The PCP verifier V_{ACSP} has the PCP explicit proof-of-knowledge property with error $\varepsilon(n) = 1/2$.*

Proof. The PCP verifier V_{ACSP} is described in Construction 8.4. The PCP weak proof-of-knowledge property of V_{ACSP} can be deduced by examining the part of the proof of Theorem 8.1 that establishes the soundness property of V_{ACSP}. Details follow.

Fix an instance $(x,1^t)$ and a proof $\pi = (p_0,\pi_1,p_1,\pi_1,\pi_{\mathsf{c}})$. Suppose that $\Pr\left[V^{\pi}_{\mathrm{ACSP}}\big((x,1^t)\big) = 1\right] > 1/2$. Let $m := \lceil \log |x| \rceil$, $S_m := \mathrm{span}(\alpha^{\mathbf{S}}_1(\mathsf{x}),\ldots,\alpha^{\mathbf{S}}_m(\mathsf{x}))$, A_x be the low-degree extension of the function $f\colon S_m \to \{0,1\}$ defined by $f(\alpha^{(x)}_i) := x_i$, and p_x the evaluation of A_x on $\mathbb{F}_t$.

It must be that

- p_0 is δ_{RS}-close to $\mathrm{RS}(\mathbb{F}_t,\mathbb{F}_t,2^{\mathsf{m}_{\mathbf{H}}(t)}-1)$,
- p_1 is δ_{VRS}-close to $\mathrm{VRS}(\mathbb{F}_t,\mathbb{F}_t,H_t,d^t_0 + (2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d^t_i)$, and
- $p_0 - p_x$ is δ_{c}-close to $\mathrm{VRS}(\mathbb{F}_t,\mathbb{F}_t,S_m,2^{\mathsf{m}_{\mathbf{H}}(t)}-1)$,

where $\delta_{\mathrm{RS}} = \delta_{\mathsf{c}} = \frac{1}{8\mathsf{c}_{\mathbf{N}}(t)}$ and $\delta_{\mathrm{VRS}} = \frac{1}{8}$. Indeed, if that were not the case, then $V^{\pi}_{\mathrm{ACSP}}\big((x,1^t)\big)$ would have accepted with probability at most $1/2$ (due respectively to its first, second, and fourth subtest).

Let $A\colon \mathbb{F}_t \to \mathbb{F}_t$ be the unique polynomial of degree less than $2^{\mathsf{m}_{\mathbf{H}}(t)}$ whose evaluation table over $\mathbb{F}_t$ is closest to p_0. Note that A is unique, because $1 - (2^{\mathsf{m}_{\mathbf{H}}(t)}-1)/|\mathbb{F}_t|$ is larger than $2\delta_{\mathrm{RS}} = 1/(4\mathsf{c}_{\mathbf{N}}(t))$, as guaranteed by Definition 7.1. (See Lemma 3.11.) Furthermore, we can deduce that A is consistent with the instance $(x,1^t)$: $p_0 - p_x$ is $1/(8\mathsf{c}_{\mathbf{N}}(t))$-close to the evaluation

of $A - A_x$ on $\mathbb{F}_t$ and $1/(8\mathsf{c}_{\mathbf{N}}(t))$-close the code $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, S_m, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$; thus, invoking once again the fact that $1 - (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)/|\mathbb{F}_t|$ is larger than $1/(4\mathsf{c}_{\mathbf{N}}(t))$ (and Lemma 3.11) we deduce that the evaluation of $A - A_x$ is in $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, S_m, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$.

Let $B\colon \mathbb{F}_t \to \mathbb{F}_t$ be the polynomial defined as $B(x) := P_t\big(x, A(\mathsf{aff}_{t,1}(x)), \ldots, A(\mathsf{aff}_{t,\mathsf{c}_{\mathbf{N}}(t)}(x))\big)$.

Define the function $p_2\colon \mathbb{F}_t \to \mathbb{F}_t$ by

$$p_2(x) := P_t\Big(x, p_0\big(\mathsf{aff}_{t,1}(x)\big), \ldots, p_0\big(\mathsf{aff}_{t,\mathsf{c}_{\mathbf{N}}(t)}(x)\big)\Big) \ .$$

Observe that, for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$ and a random $\alpha(\mathsf{x})$ in $\mathbb{F}_t$, since $\mathsf{aff}_{t,i}$ is an affine function, $\mathsf{aff}_{t,i}(\alpha(\mathsf{x}))$ is a random element in $\mathbb{F}_t$. We deduce then, via a union bound (on the fact that p_0 is $\frac{1}{8\mathsf{c}_{\mathbf{N}}(t)}$-close to the evaluation table of A over $\mathbb{F}_t$), that p_2 must be $\frac{1}{8}$-close to the evaluation table of B over $\mathbb{F}_t$.

Now let $B'\colon \mathbb{F}_t \to \mathbb{F}_t$ be the unique polynomial of degree at most $d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t$ vanishing on H_t whose evaluation table over $\mathbb{F}_t$ is closest to p_1. Again note that B' is unique, because $1 - (d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t)/|\mathbb{F}_t|$ is larger than $2\delta_{\mathrm{VRS}} = 1/4$, as guaranteed by Definition 7.1. (See Lemma 3.11.) We argue that B' and B must be equal. Indeed, suppose by way of contradiction that B and B' are distinct; then, as they are both polynomials of degree at most $d_0^t + (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} d_i^t \leq |\mathbb{F}_t|/4$, their evaluation tables over $\mathbb{F}_t$ may agree on at most a $1/4$ fraction of their entries; hence, by a union bound, the third subtest of V_{ACSP} accepts with probability at most $1/8 + 1/8 + 1/4 \leq 1/2$ — a contradiction to the fact that $V^{\pi}_{\mathrm{ACSP}}((x, 1^t))$ accepts with probability greater than $1/2$.

Since B' and B are equal, B vanishes on H_t, so that (together with the fact that A is consistent with the instance $(x, 1^t)$) we deduce that A is a witness to the fact that $(x, 1^t) \in$ SuccinctACSP.

In summary, p_0 a function over $\mathbb{F}_t$ that is $\frac{1}{8\mathsf{c}_{\mathbf{N}}(t)}$-close to a (unique) polynomial A of degree less than $2^{\mathsf{m}_{\mathbf{H}}(t)}$; moreover A is a witness to the fact that $(x, 1^t) \in$ SuccinctACSP. Therefore, we can let E_{PCP} be the algorithm that decodes the Reed-Solomon codeword p_0. More precisely, by Claim 3.12, there exists a polynomial-time algorithm DecodeRS that, on input (the representation of) a finite field $\mathbb{F}$, a subset S of $\mathbb{F}$, a degree d (presented in unary), and a function $p\colon S \to \mathbb{F}$, outputs a polynomial $P\colon \mathbb{F} \to \mathbb{F}$ of degree at most d whose evaluation over S is closest to p (as long as p lies in the unique decoding radius). We can then define E_{PCP} as follows:

$E^{\pi}_{\mathrm{PCP}}\big(x, 1^{2^t}\big) \stackrel{\mathrm{def}}{=}$

1. Parse π as (p_0, π_0, p_1, π_1).
2. Compute $I_t := \textsc{FindIrrPoly}\big(1^{f(t)}\big)$.
3. Set $\mathcal{B}_{\mathbb{F}_t} := \big(1, \mathsf{x}, \ldots, \mathsf{x}^{f(t)-1}\big)$.
4. Set $d := 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1$.
5. Compute $A(x) := \textsc{DecodeRS}(I_t, \mathrm{span}(\mathcal{B}_{\mathbb{F}_t}), 1^d, p_0)$.
6. Output $A(x)$.

Note that, similarly to the verifier V_{ACSP}, the knowledge extractor E_{PCP} also knows the choice of parameters for SuccinctACSP; so, for example, it knows the functions f and $\mathsf{m}_{\mathbf{H}}(t)$. Clearly, E_{PCP} runs in polynomial time.[20] Moreover, it is also easy to see that, by construction, whenever

[20] More precisely, E_{PCP} will run in $\mathsf{poly}(2^{\mathsf{m}_{\mathbf{H}}(t)})$ time, which is polynomial in 1^{2^t} whenever $\mathsf{m}_{\mathbf{H}}(t) = t + o(t)$. Indeed, the "true" measure of instance size in SuccinctACSP problems is $2^{\mathsf{m}_{\mathbf{H}}(t)}$ and not 2^t.

V_{ACSP} accepts the instance $(x, 1^t)$ and oracle π with probability greater than $1/2$, it is indeed the case that E_{PCP} succeeds in extracting a witness with probability greater than $2/3$. (In fact, it succeeds with probability 1.) □

Remark 10.37. Even for an "honest" oracle $\pi = (p_0, \pi_0, p_1, \pi_1, \pi_{\mathsf{c}})$ (e.g., the one produced by the PCP prover P_{ACSP} on input $(x, 1^t)$ and a witness A for $(x, 1^t)$), retrieving A from its error-free evaluation p_0 over $\mathbb{F}_t$ requires working at least linearly in $\deg(A)$, which may very well be linear in 2^t. Indeed, simply reading less than $\deg(A)$ values of p_0 is not enough information to uniquely determine A. Hence, at least for V_{ACSP} we must settle for a "weak" proof-of-knowledge where if we want to extract a bit of the witness, then we might as well extract the whole witness. Fortunately, as discussed in more detail in Section 10.6, for many applications of universal arguments this is not an issue.

Remark 10.38. Fix $\kappa \in \mathbb{N}$. If a PCP verifier V has the PCP explicit proof-of-knowledge property with error ε, then the PCP verifier V sequentially repeated κ times has the PCP explicit proof-of-knowledge property with error ε^κ. (Indeed, whenever the repeated verifier accepts a PCP oracle π with probability greater than $2^{-\kappa}$, then the non-repeated verifier must accept the same PCP oracle π with greater than half probability.)

Remark 10.39. Barak and Goldreich claimed [BG02, Appendix A] that the PCP proof of knowledge holds for many PCPs, yet the PCPs that we use only satisfy the explicit PCP proof of knowledge. The reason is that almost all other PCP constructions (e.g., [BFLS91]) use *multivariate* techniques, as opposed to *univariate* techniques as we do. More precisely, other PCP constructions use low-degree low-variate multivariate polynomials when it comes to arithmetizing certain constraint satisfaction problems. With multivariate techniques, the witness is encoded using a Reed-Muller code (via a low-degree extension), which has much better local-decoding properties than Reed-Solomon codes: a witness w of size 2^t is mapped to a d-degree m-variate polynomial with $m = O(t/\log t)$ and $d = O(t)$ and a bit of w can be retrieved in time $\mathsf{poly}(t)$ [BF90][Lip90].

Of course, the advantage of univariate techniques over multivariate techniques is that the proof length of the PCP oracle is much shorter. (Namely, we do not know of PCPs of quasilinear length based on multivariate techniques.)

10.6 Obtaining (Almost) Universal Arguments

Having proved that the PCP system $(P_{\text{ACSP}}, V_{\text{ACSP}})$ for SuccinctACSP does satisfy the first three properties and a variant of the fourth property considered by Barak and Goldreich [BG02, Definiton 3.2], we now explain how these four properties are enough to imply, through [BG02, Construction 3.4], an "almost universal argument system" for SuccinctACSP. (See Definition 10.9.)

Claim 10.40. *Following [BG02, Construction 3.4] starting with the (amplified) PCP system $(P_{\text{ACSP}}, V_{\text{ACSP}})$ yields an almost universal argument system $(P_{\mathsf{UA}}, V_{\mathsf{UA}})$ for the language* SuccinctACSP. *(Of course, by invoking the construction with a collision-resistant function family as well.)*

We assume familiarity with the proof of [BG02, Lemma 3.5], which shows that the construction of universal arguments given in [BG02, Construction 3.4] using a PCP system satisfying the

original four properties in [BG02, Definition 3.2] works, as long as a collision-resistant function family is used.

In comparison, our Claim 10.40 says that, even if we start with a PCP system that satisfies a slightly weaker notion of proof of knowledge (namely, Definition 10.8 instead of Definition 10.6), the construction of universal arguments still yields a universal argument, albeit with a correspondingly weaker notion of proof of knowledge, which we call an *almost universal argument*. (And, of course, the other two properties of efficient verification and completeness via a relatively-efficient prover easily follow from using the same construction [BG02, Construction 3.4].)

As discussed in Remark 10.43, this is sufficient for many applications of universal arguments.

Proof. We give a high-level overview of the proof [BG02, Lemma 3.5], and then explain (and give details of) how it can be modified to yield the proof for our claim. Also, in this proof we assume that V_{ACSP} has been amplified to soundness $2^{-\kappa}$ for inputs of length κ; in particular, it has the PCP explicit proof-of-knowledge property with error $\varepsilon(\kappa) = 2^{-\kappa}$. (See Remark 10.38.)

Roughly, the argument used to prove [BG02, Lemma 3.5] consists of three steps:

1. First, one shows that, if a prover circuit is too inconsistent about answering queries and yet is still somewhat convincing, then that prover can be efficiently converted into a collision-finding circuit; in particular, that means that, in order to be somewhat convincing, a prover circuit must essentially "have a PCP oracle in mind".

2. Next, one shows how to construct an efficient "oracle recovery" procedure, i.e., a probabilistic polynomial-time oracle machine $\mathsf{recover}$ that, with access to the prover circuit, is an implicit representation of a PCP oracle that is also somewhat convincing (this time convincing to the PCP verifier).

3. Finally, one shows how this oracle recovery procedure can be used, together with the PCP knowledge extractor, to yield the knowledge extractor required by the weak proof-of-knowledge property from Definition 10.6.

Even given this high-level overview, it is clear that the first two steps of the argument of Barak and Goldreich work in our case too, because they only rely on the collision-resistant property and the fact that the prover is somewhat convincing. On the other hand, step three (which clearly refers to the PCP knowledge extractor), requires modification. So we now show how to use the oracle recovery procedure together with our knowledge extractor to produce a "weaker" universal-argument knowledge extractor that will satisfy Definition 10.8 instead.

More precisely, we argue as follows. Fix two positive polynomials s_{UA} and p_{UA} and, letting $q_{\mathsf{UA}} = 5p_{\mathsf{UA}}$, we show how to construct a probabilistic polynomial-time procedure E_{UA} (depending on s_{UA} and p_{UA}) such that, for every family of s_{UA}-size prover circuits $\widetilde{P}_{\mathsf{UA}} = \{\widetilde{P}_{\mathsf{UA},\kappa}\}_{\kappa\in\mathbb{N}}$, for all sufficiently large $\kappa \in \mathbb{N}$, for every instance $(x, 1^t) \in \{0,1\}^\kappa$, if $\widetilde{P}_{\mathsf{UA},\kappa}$ convinces V_{UA} to accept $(x, 1^t)$ with probability greater than $p_{\mathsf{UA}}(\kappa)^{-1}$ then, with probability greater than $q_{\mathsf{UA}}(\kappa)^{-1}$ taken over a random choice of internal randomness r for E_{UA}, the weak knowledge extractor E_{UA}, with oracle access to the code of $\widetilde{P}_{\mathsf{UA},\kappa}$ and on input $(x, 1^t)$, outputs a valid witness w for x.

Barak and Goldreich [BG02, Claim 3.5.3] showed how to construct a probabilistic polynomial-time oracle machine $\mathsf{recover}$ (depending on s_{UA} and p_{UA}) such that, for all sufficiently large $\kappa \in \mathbb{N}$, if $\widetilde{P}_{\mathsf{UA},\kappa}$ convinces V_{UA} to accept $(x, 1^t)$ with probability greater than $p_{\mathsf{UA}}(\kappa)^{-1}$, then, with proba-

bility $\frac{1}{4.5p_{\mathsf{UA}}(\kappa)}$, $\mathsf{recover}^{\langle \widetilde{P}_{\mathsf{UA},\kappa}\rangle}\big((x,1^t),\cdot\big)$ is an implicit representation of a PCP oracle π such that $\Pr\left[V^{\pi}_{\mathrm{ACSP}}\big((x,1^t)\big)=1\right] > \frac{1}{8p_{\mathsf{UA}}(\kappa)} > \frac{1}{2^\kappa}$.[21]

Also, we amplify the success probability of $E_{\mathrm{PCP}}(x,1^{2^t})$, with $(x,1^t)\in\{0,1\}^\kappa$, from greater than $2/3$ to greater than $1-2^{-\kappa}$.

Then, we define the "explicit" weak knowledge extractor E_{UA} (depending on s_{UA} and p_{UA}) as follows: for every family of s_{UA}-size prover circuits $\widetilde{P}_{\mathsf{UA}} = \{\widetilde{P}_{\mathsf{UA},\kappa}\}_{\kappa\in\mathbb{N}}$, for every $\kappa\in\mathbb{N}$ and instance $(x,1^t)\in\{0,1\}^\kappa$,

$E^{\langle \widetilde{P}_{\mathsf{UA},\kappa}\rangle}_{\mathsf{UA}}(x,1^{2^t}) \stackrel{\mathrm{def}}{=}$

1. Draw a random tape ω for the oracle-recovery procedure $\mathsf{recover}$.
2. Invoke $E_{\mathrm{PCP}}(x,1^{2^t})$ providing it with oracle access to π, using the oracle-recovery procedure $\mathsf{recover}$ on input (x,t), random tape ω, and oracle access to the code of $\widetilde{P}_{\mathsf{UA},\kappa}$.
3. Output whatever (the amplified) $E^{\pi}_{\mathrm{PCP}}(x,1^{2^t})$ outputs.

Note that E_{UA} does run in polynomial time, because each invocation of $\mathsf{recover}$ requires $\mathsf{poly}(\kappa)$ invocations of $\widetilde{P}_{\mathsf{UA},\kappa}$ (for a total of $\mathsf{poly}(\kappa)\cdot s_{\mathsf{UA}}(\kappa)$ time per invocation), and there are at most $\mathsf{poly}(|y|,t,\kappa)$ invocations of $\mathsf{recover}_\varepsilon$ (as the amplified E_{PCP} runs in $\mathsf{poly}(|y|,t,\kappa)$ time).

Finally, the probability that $E^{\langle \widetilde{P}_{\mathsf{UA},\kappa}\rangle}_{\mathsf{UA}}(x,1^{2^t})$ succeeds in extracting a valid witness for y is at least the probability that ω is a "good" random tape for $\mathsf{recover}\big((x,t)\big)$ (i.e., makes it an implicit representation of a "convincing" PCP oracle π) times the probability that $E^{\pi}_{\mathrm{PCP}}(x,1^{2^t})$ successfully extracts; for sufficiently large $\kappa\in\mathbb{N}$, that probability is at least $\frac{1}{4.5p_{\mathsf{UA}}(\kappa)}\cdot(1-2^{-\kappa}) > \frac{1}{5p_{\mathsf{UA}}(\kappa)}$, as desired. □

Remark 10.41. Note that the alphabet of the PCP oracle is elements of a finite field $\mathbb{F}_{2^\ell}$, and not bits. Fortunately, however, the analysis of the rewinding strategy of the knowledge extractor does not rely on how large the alphabet of the PCP oracle is, but only reasons about consistency of answers across different runs of the prover, and thus the different alphabet does not cause any changes in the proof.

Remark 10.42. Even probabilistically checkable *arguments* (PCAs) suffice for the construction of universal arguments (including almost universal arguments), as long as they satisfy analogous knowledge properties. Indeed, both the proof of [BG02, Lemma 3.5] and our proof of Claim 10.40 would go through even if the guarantees of the PCP knowledge extractor E_{PCP} were to hold only for PCP oracles π generated by efficient procedures. (And this restriction to efficient procedures is not "in addition" as we are already using collision-resistant function families.)

In principle, PCAs could be easier to construct, and thus closer to practicality. The authors give an example of a PCA in [BSCGT12], where *computational* Levin reductions simplify the "structuring" of a computation transcript, by avoiding routing techniques altogether.

Remark 10.43. In many (especially positive) applications of universal arguments, one considers $(x,1^t)$ where 2^t is bounded by some polynomial in $|x|$. For example, a simple and direct use of universal arguments as a way to delegate computation (possibly with the trivial composition

[21] Actually, the oracle-recovery procedure of Barak and Goldreich was constructed to take a randomly chosen collision-resistant function family seed α as input, but here we include this random choice as part of the procedure.

with a semantically-secure fully-homomorphic encryption to ensure privacy), or as a building block to other cryptographic primitives (e.g., [CT10] or [BCCT11]).

A Other Related Work

Are PCPs needed for fast verification of long computations? While we believe that the question of whether PCPs can be made efficient enough to be useful in practice is itself fascinating, a natural question to consider is whether the "full machinery" of PCPs is really needed for the applications that one has in mind.

The answer seems to depend on the strength of the desired application and the strength of the computational assumptions that one is willing to make. Let us explain.

For example, if one is only interested in verifying deterministic computations (as opposed to verifying non-deterministic ones) and also tolerate an expensive offline preprocessing phase, then there are constructions that avoid the use of PCPs [GGP10, CKV10, AIK10]. (Though, admittedly, these works ultimately rely on "the other heavy hammer" of fully-homomorphic encryption, which, just like PCPs, is also notoriously inefficient.)

However, the aforementioned works deliver a notion of soundness that is not so strong: soundness only holds when the information of whether the verifier has accepted or not remains secret. One may wonder whether such a defect is brought about by a "lack of PCPs".

In a sense, this is *not* the case:

- Following the work of Ishai et al. [IKO07], Ishai and Ostrovsky [IO11] showed that, under a natural (but non-falsifiable) assumption about encryption schemes, one can use simple PCPs (based on the Hadamard code) to construct designated-verifier succinct non-interactive arguments (SNARGs) with preprocessing that remain secure even in the presence of a verification oracle. While the techniques of Ishai and Ostrovsky do not invoke the full machinery of PCPs (but only simple probabilistic-checking techniques based on the Hadamard code) and obtain an online verification phase that is very fast, their techniques seem to inherently suffer from a quadratic blowup in the prover running time and preprocessing running time.
- Groth [Gro10] (essentially) showed that, under a knowledge-of-exponent assumption in bilinear groups, one can construct publicly-verifiable SNARGs with preprocessing. While the techniques of Groth do not explicitly invoke probabilistic-checking techniques (though may ultimately implicitly do) and also obtain a very fast only verification phase, his techniques have similar drawbacks as the ones of Ishai and Ostrvosky [IO11]: they also suffer from a quadratic blowup in the prover running time and preprocessing running time. (Lipmaa [Lip12] recently improved on the work of Groth [Gro10] by showing how to make the preprocessing running time quasilinear; however, the prover running time remains quadratic.)

One may wonder again whether suffering from a preprocessing phase is due a lack of "full-fledged" PCPs.

Surprisingly, the answer is again *no*: Bitansky et al. [BCCT12] showed that, as long as a SNARG satisfies a proof of knowledge property, then the preprocessing phase can always be removed (by only additionally invoking standard cryptographic assumptions).

Thus, it seems that the question of whether "full-fledged" PCPs are really needed depends on the strength of the computational assumptions we are willing to make. On the one hand, we could make strong non-standard assumptions to justify either [IO11] or [Gro10] and then invoke the result of [BCCT12] to obtain succinct argument constructions that do not invoke "full-fledged" PCPs; on the other hand, we could only assume the existence of collision-resistant hash functions and construct succinct arguments using "full-fledged" PCPs [Kil92, BG02].

B Complexity Analysis

We analyze the complexity of everyalgorithm. Concretely, for each "prover-verifier pair", we carefully analyze the query complexity, randomness complexity, and proof length complexity.[22] The resulting expressions can be easily evaluated numerically for any given set of parameters.

Throughout this section, we fix a choice of integer parameters $(\eta, \kappa_0, \gamma, \mu)$ satisfying Equation 7.

B.1 Complexity Analysis of $P_{\mathrm{RS},=}$ and $V_{\mathrm{RS},=}$

Lemma B.1 (Complexity Parameters for $P_{\mathrm{RS},=}$ and $V_{\mathrm{RS},=}$)**.** *The following equations hold:*

$$\begin{aligned} \mathsf{query}_{\mathrm{RS},=}(\ell,\kappa) &\leq 2^{\kappa_0} \ , \\ \mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa) &\leq \kappa + (\mu+2)\log(2\kappa) \ , \\ \mathsf{length}_{\mathrm{RS},=}(\ell,\kappa) &\leq 2^{\kappa}\kappa^{1+\max\{-\gamma+\mu+1,\gamma\}} \ . \end{aligned}$$

Proof. See Section 10.3.1.

First, as was already remarked in [BSS08, Propistion 6.8], it is easy to see that the query complexity is as follows:

$$\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa) = \begin{Bmatrix} 2^{\kappa} & \text{if } \kappa \leq \kappa_0 \\ 0 & \text{if } \kappa > \kappa_0 \end{Bmatrix} \leq 2^{\kappa_0} \ .$$

Next, as was already remarked in [BSS08, Proposition 6.8] (and then in further detail in [Bha05, Sec. 2.3.1]), the randomness complexity satisfies the recursion from Lemma B.9 (after appropriately generalizing it to our more general constructions), and the lemma tells us that we can upper bound it follows:

$$\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa) \leq \begin{Bmatrix} 0 & \text{if } \kappa \leq \kappa_0 \\ \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma+\mu+1,\gamma\}\right) & \text{if } \kappa > \kappa_0 \end{Bmatrix} \leq \kappa + (\mu+2)\log(2\kappa) \ .$$

Finally, as was already remarked in [BSS08, Proposition 6.8] (and then in further detail in [Bha05, Sec. 2.2.2]), the proof length complexity satisfies the recursion from Lemma B.10 (after appropriately generalizing it to our more general constructions), and the lemma tells us that we can upper bound it as follows:

$$\mathsf{length}_{\mathrm{RS},=}(\ell,\kappa) \leq 2^{\kappa}\kappa^{1+\max\{-\gamma+\mu+1,\gamma\}} \ .$$

Of course, if one wishes to numerically evaluate $\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)$ or $\mathsf{length}_{\mathrm{RS},=}(\ell,\kappa)$, using the recursive function (in Lemma B.9 or Lemma B.10 respectively) will give the exact value. (Though our bounds are quite tight.) □

[22] We do not discuss here the prover and verifier time and space complexities; indeed, a detailed complexity analysis of such performance measures (e.g., by paying close attention to multiplicative constants) does not seem feasible and is ultimately not fruitful — after all, concrete time and space requirements of a prover and verifier are best studied through a code implementation, which we leave to future work.

B.2 Complexity Analysis of $P_{\mathrm{RS},<}$ and $V_{\mathrm{RS},<}$

Lemma B.2 (Complexity Parameters for $P_{\mathrm{RS},<}$ and $V_{\mathrm{RS},<}$)**.** *The following equations hold:*

$$\begin{aligned}
\mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d) &= 2\cdot\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)\ ,\\
\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d) &= \mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)\ ,\\
\mathsf{length}_{\mathrm{RS},<}(\ell,\kappa,d) &= 2\cdot\mathsf{length}_{\mathrm{RS},=}(\ell,\kappa)\ .
\end{aligned}$$

Proof. See Section 10.3.2. □

B.3 Complexity Analysis of $P_{\mathrm{RS},>}$ and $V_{\mathrm{RS},>}$

Lemma B.3 (Complexity Parameters for $P_{\mathrm{RS},>}$ and $V_{\mathrm{RS},>}$)**.** *The following equations hold:*

$$\begin{aligned}
\mathsf{query}_{\mathrm{RS},>}(\ell,\kappa,d) &= m_{\kappa,\eta,d}\cdot\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)+\mathbf{1}_{\kappa,\eta,d}\cdot\mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d-m_{\kappa,\eta,d}(d_{\kappa,\eta}+1))+m_{\kappa,\eta,d}+\mathbf{1}_{\kappa,\eta,d}+1\ ,\\
\mathsf{rand}_{\mathrm{RS},>}(\ell,\kappa,d) &= \max\left\{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)\,,\ \mathbf{1}_{\kappa,\eta,d}\cdot\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d-m_{\kappa,\eta,d}(d_{\kappa,\eta}+1))\,,\ \kappa\right\}\ ,\\
\mathsf{length}_{\mathrm{RS},>}(\ell,\kappa,d) &= m_{\kappa,\eta,d}\cdot(2^{\kappa}+\mathsf{length}_{\mathrm{RS},=}(\ell,\kappa))+\mathbf{1}_{\kappa,\eta,d}\cdot(2^{\kappa}+\mathsf{length}_{\mathrm{RS},<}(\ell,\kappa,d-m_{\kappa,\eta,d}(d_{\kappa,\eta}+1)))\ ,
\end{aligned}$$

where $m_{\kappa,\eta,d}:=\lceil\frac{d+1}{d_{\kappa,\eta}+1}\rceil$ *and* $\mathbf{1}_{\kappa,\eta,d}:=1$ *if* $(d+1)>m_{\kappa,\eta,d}\cdot(d_{\kappa,\eta}+1)$ *and* $\mathbf{1}_{\kappa,\eta,d}:=0$ *otherwise.*

Proof. See Section 10.3.3. □

B.4 Complexity Analysis of P_{RS} and V_{RS}

Lemma B.4 (Complexity Parameters for P_{RS} and V_{RS})**.** *The following equations hold:*

$$\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)=\begin{cases}\mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d) & \text{if } d<d_{\kappa,\eta}\\ \mathsf{query}_{\mathrm{RS},=}(\ell,\kappa) & \text{if } d=d_{\kappa,\eta}\\ \mathsf{query}_{\mathrm{RS},>}(\ell,\kappa,d) & \text{if } d>d_{\kappa,\eta}\end{cases},$$

$$\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)=\begin{cases}\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d) & \text{if } d<d_{\kappa,\eta}\\ \mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa) & \text{if } d=d_{\kappa,\eta}\\ \mathsf{rand}_{\mathrm{RS},>}(\ell,\kappa,d) & \text{if } d>d_{\kappa,\eta}\end{cases},$$

$$\mathsf{length}_{\mathrm{RS}}(\ell,\kappa,d)=\begin{cases}\mathsf{length}_{\mathrm{RS},<}(\ell,\kappa,d) & \text{if } d<d_{\kappa,\eta}\\ \mathsf{length}_{\mathrm{RS},=}(\ell,\kappa) & \text{if } d=d_{\kappa,\eta}\\ \mathsf{length}_{\mathrm{RS},>}(\ell,\kappa,d) & \text{if } d>d_{\kappa,\eta}\end{cases}.$$

Proof. See Section 10.3.4.

The prover P_{RS} simply calls $P_{\mathrm{RS},<}$, $P_{\mathrm{RS},=}$, or $P_{\mathrm{RS},>}$, depending on whether the input degree d is less than, equal to, or greater than $d_{\kappa,\eta}:=2^{\kappa}/2^{\eta}-1$; similarly for V_{RS}.[23] □

[23] Though if κ no larger than η, then V_{RS} will directly test the degree of the implicit input. We do not include this case in the complexity analysis, because η is a very small constant, and we will never be interested in inputs of such a small dimension!

B.5 Complexity Analysis of P_{VRS} and V_{VRS}

Lemma B.5 (Complexity Parameters for P_{VRS} and V_{VRS})**.** *The following equations hold:*

$$\begin{aligned}
\mathsf{query}_{\text{VRS}}(\ell, \kappa, \lambda, d) &= \mathsf{query}_{\text{RS}}(\ell, \kappa, d - 2^{\lambda}) + 2 \ , \\
\mathsf{rand}_{\text{VRS}}(\ell, \kappa, \lambda, d) &= \max\left\{\mathsf{rand}_{\text{RS}}(\ell, \kappa, d - 2^{\lambda}), \kappa\right\} \ , \\
\mathsf{length}_{\text{VRS}}(\ell, \kappa, \lambda, d) &= 2^{\kappa} + \mathsf{length}_{\text{RS}}(\ell, \kappa, d - 2^{\lambda}) \ ,
\end{aligned}$$

Proof. See Section 10.3.5. □

B.6 Complexity Analysis of V_{aRS} and V_{aVRS}

Lemma B.6 (Complexity Parameters for V_{aRS})**.** *The following equations hold:*

$$\begin{aligned}
\mathsf{query}_{\text{aRS}}(\ell, \kappa, d, \delta, s') &= m_{s_{\text{RS}}, \delta, s'}(n) \cdot \mathsf{query}_{\text{RS}}(\ell, \kappa, d) \ , \\
\mathsf{rand}_{\text{aRS}}(\ell, \kappa, d, \delta, s') &= m_{s_{\text{RS}}, \delta, s'}(n) \cdot \mathsf{rand}_{\text{RS}}(\ell, \kappa, d) \ .
\end{aligned}$$

Proof. See Section 10.3.6.

Observe that V_{aRS} simply performs naive sequential repetition on V_{RS}. Hence, by Remark 6.2, in order to obtain a "weak" PCPP with proximity parameter δ and target soundness s' (both given as input to V_{aRS}), the number of repetitions is:

$$m_{s_{\text{RS}}, \delta, s'}(n) = \left\lceil \frac{\log(1 - s')}{s_{\text{RS}}(\delta, n)} \right\rceil \ .$$

where $n = 2^{\kappa}$ and s_{RS} is the soundness of V_{RS}. (Recall that Theorem 6.1 gives a lower bound on s_{RS}.)

The query complexity and randomness complexity thus simply increase by the multiplicative factor $m_{s_{\text{RS}}, \delta, s'}(n)$. □

Lemma B.7 (Complexity Parameters for V_{aVRS})**.** *The following equations hold:*

$$\begin{aligned}
\mathsf{query}_{\text{aVRS}}(\ell, \kappa, \lambda, d, \delta, s') &= m_{s_{\text{VRS}}, \delta, s'}(n) \cdot \mathsf{query}_{\text{VRS}}(\ell, \kappa, \lambda, d) \ , \\
\mathsf{rand}_{\text{aVRS}}(\ell, \kappa, \lambda, d, \delta, s') &= m_{s_{\text{VRS}}, \delta, s'}(n) \cdot \mathsf{rand}_{\text{VRS}}(\ell, \kappa, \lambda, d) \ .
\end{aligned}$$

Proof. See Section 10.3.7.

Observe that V_{aVRS} simply performs naive sequential repetition on V_{VRS}. Hence, by Remark 6.2, in order to obtain a "weak" PCPP with proximity parameter δ and target soundness s' (both given as input to V_{aVRS}), the number of repetitions is:

$$m_{s_{\text{VRS}}, \delta, s'}(n) = \left\lceil \frac{\log(1 - s')}{s_{\text{VRS}}(\delta, n)} \right\rceil \ .$$

where $n = 2^{\kappa}$ and s_{VRS} is the soundness of V_{VRS}. (Recall that Theorem 6.1 gives a lower bound on s_{VRS}.)

The query complexity and randomness complexity thus simply increase by the multiplicative factor $m_{s_{\text{VRS}}, \delta, s'}(n)$. □

B.7 Complexity Analysis of P_{ACSP} and V_{ACSP}

Lemma B.8 (Complexity Parameters for P_{ACSP} and V_{ACSP})**.** *The following equations hold:*

$$\begin{aligned}
\mathsf{query}_{\text{ACSP}}(x,t) = {} & \mathsf{query}_{\text{aRS}}(f(t), f(t), 2^{\mathsf{m_H}(t)} - 1, \tfrac{1}{8\mathsf{c_N}(t)}, \tfrac{1}{2}) \\
& + \mathsf{query}_{\text{aVRS}}(f(t), f(t), \mathsf{m_H}(t), d_0^t + (2^{\mathsf{m_H}(t)} - 1)\textstyle\sum_{i=1}^{\mathsf{c_N}(t)} d_i^t, \tfrac{1}{8}, \tfrac{1}{2}) \\
& + (\mathsf{c_N}(t) + 1) \\
& + \mathsf{query}_{\text{aVRS}}(f(t), f(t), m, 2^{\mathsf{m_H}(t)} - 1, \tfrac{1}{8\mathsf{c_N}(t)}, \tfrac{1}{2}) \ , \\
\mathsf{rand}_{\text{ACSP}}(x,t) = {} & \max\Big\{\mathsf{rand}_{\text{aRS}}(f(t), f(t), 2^{\mathsf{m_H}(t)} - 1, \tfrac{1}{8\mathsf{c_N}(t)}, \tfrac{1}{2}), \\
& \qquad \mathsf{rand}_{\text{aVRS}}(f(t), f(t), \mathsf{m_H}(t), d_0^t + (2^{\mathsf{m_H}(t)} - 1)\textstyle\sum_{i=1}^{\mathsf{c_N}(t)} d_i^t, \tfrac{1}{8}, \tfrac{1}{2}), \\
& \qquad f(t), \\
& \qquad \mathsf{rand}_{\text{aVRS}}(f(t), f(t), m, 2^{\mathsf{m_H}(t)} - 1, \tfrac{1}{8\mathsf{c_N}(t)}, \tfrac{1}{2})\Big\} \ , \\
\mathsf{length}_{\text{ACSP}}(x,t) = {} & 2^{f(t)} + \mathsf{length}_{\text{aRS}}(f(t), f(t), 2^{\mathsf{m_H}(t)} - 1, \tfrac{1}{8\mathsf{c_N}(t)}, \tfrac{1}{2}) \\
& + 2^{f(t)} + \mathsf{length}_{\text{aVRS}}(f(t), f(t), \mathsf{m_H}(t), d_0^t + (2^{\mathsf{m_H}(t)} - 1)\textstyle\sum_{i=1}^{\mathsf{c_N}(t)} d_i^t, \tfrac{1}{8}, \tfrac{1}{2}) \\
& + \mathsf{length}_{\text{aVRS}}(f(t), f(t), m, 2^{\mathsf{m_H}(t)} - 1, \tfrac{1}{8\mathsf{c_N}(t)}, \tfrac{1}{2}) \ ,
\end{aligned}$$

where $m := \lceil \log |x| \rceil$.

Proof. See Section 10.4.8. The equations above can be verified by inspection. □

B.8 Solving Recursions

We deduce an upper bound for two recursive functions that arise in Appendix B.1: respectively, the upper bounds are on the recursive function giving the amount of randomness used by $V_{\text{RS},=}$ (Lemma B.9) and the recursive function giving the length of the proximity proof generated by $P_{\text{RS},=}$ (Lemma B.10).

Lemma B.9. *The recursion*

$$r(\kappa) = \begin{cases} 0 & \text{if } \kappa \le \kappa_0 \\ 1 + \max\left\{\left\lceil \frac{\kappa}{2} \right\rceil + \gamma + r\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1\right), \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + r\left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma\right)\right\} & \text{if } \kappa > \kappa_0 \end{cases} \tag{28}$$

is upper bounded by the function

$$\widetilde{r}(\kappa) = \begin{cases} 0 & \text{if } \kappa \le \kappa_0 \\ \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma + \mu + 1, \gamma\}\right) & \text{if } \kappa > \kappa_0 \end{cases} .$$

Proof. For $\kappa \in \{1, \ldots, \kappa_0\}$, the equality follows immediately. The parameter constraints from Equation 7 imply that $\widetilde{r}(k)$ is well-defined for $\kappa > \kappa_0$:

$$\begin{aligned}
&\frac{k}{2} \ge \frac{\kappa_0 + 1}{2} \ge \left\lfloor \frac{\kappa_0 + 1}{2} \right\rfloor \ge \gamma + 1 \ge \gamma + \frac{1}{2} \quad \text{and} \\
&\frac{k}{2} \ge \frac{\kappa_0 + 1}{2} \ge \left\lceil \frac{\kappa_0 + 1}{2} \right\rceil - \frac{1}{2} \ge -\gamma + \mu + 2 - \frac{1}{2} = -\gamma + \mu + 1 + \frac{1}{2} \ ,
\end{aligned}$$

so that $\kappa - 2 \cdot \max\{-\gamma + \mu + 1, \gamma\} \geq 1 > 0$. Also, it will be useful to also derive the following inequalities (also implied by Equation 7):

$$\kappa - \left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma + 1\right) = \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma - 1 \geq \left\lfloor \frac{\kappa_0 + 1}{2} \right\rfloor - \gamma - 1 \geq 0 \quad \text{and}$$
$$\kappa - \left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1\right) = \left\lceil \frac{\kappa}{2} \right\rceil + \gamma - \mu - 1 \geq 1 \geq 0 \ .$$

Finally, we observe that, for every two κ and c with $\kappa \geq 2c$:

$$\log\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - c\right) \leq \log\left(\frac{\kappa}{2} - c\right) = \log(\kappa - 2c) - 1 \quad \text{and}$$
$$\log\left(\left\lceil \frac{\kappa}{2} \right\rceil - c\right) \leq \log\left(\frac{\kappa}{2} - c + 1\right) = \log(\kappa - 2c + 2) - 1 \ .$$

Now we go back to the rest of the proof. For $\kappa > \kappa_0$, we distinguish between four cases:

- *Case 1: κ is such that $\lfloor \kappa/2 \rfloor - \gamma + \mu + 1 \leq \kappa_0$ and $\lceil \kappa/2 \rceil + \gamma \leq \kappa_0$.* By Equation 28, the defining equation for $r(\kappa)$, we get:

$$\begin{aligned} r(\kappa) &= 1 + \max\left\{\left\lceil \frac{\kappa}{2} \right\rceil + \gamma + r\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1\right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + r\left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma\right)\right\} \\ &= 1 + \max\left\{\left\lceil \frac{\kappa}{2} \right\rceil + \gamma + 0 \, , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 0\right\} \\ &\leq \kappa \\ &\leq \kappa + (\mu + 2)\log\left(\kappa - 2\max\{-\gamma + \mu + 1, \gamma\}\right) \ . \end{aligned}$$

- *Case 2: κ is such that $\lfloor \kappa/2 \rfloor - \gamma + \mu + 1 > \kappa_0$ and $\lceil \kappa/2 \rceil + \gamma \leq \kappa_0$.* By Equation 28, the defining equation for $r(\kappa)$, we get:

$$\begin{aligned} r(\kappa) &= 1 + \max\left\{\left\lceil \frac{\kappa}{2} \right\rceil + \gamma + r\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1\right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + r\left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma\right)\right\} \\ &\leq 1 + \max\left\{\left\lceil \frac{\kappa}{2} \right\rceil + \gamma + \left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 + (\mu + 2)\log\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 - 2\max\{-\gamma + \mu + 1, \gamma\}\right)\right) \ , \right. \\ &\qquad \left. \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 0\right\} \\ &\leq 1 + \max\left\{\kappa + \mu + 1 + (\mu + 2)\log\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 - 2\max\{-\gamma + \mu + 1, \gamma\}\right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu\right\} \\ &\leq 1 + \max\left\{\kappa + \mu + 1 + (\mu + 2)\log\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \max\{-\gamma + \mu + 1, \gamma\}\right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu\right\} \\ &\leq \max\left\{\kappa + (\mu + 2)\log\left(\kappa - 2\max\{-\gamma + \mu + 1, \gamma\}\right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1\right\} \\ &= \kappa + (\mu + 2)\log\left(\kappa - 2\max\{-\gamma + \mu + 1, \gamma\}\right) \ . \end{aligned}$$

- *Case 3: κ is such that $\lfloor \kappa/2 \rfloor - \gamma + \mu + 1 \leq \kappa_0$ and $\lceil \kappa/2 \rceil + \gamma > \kappa_0$.* By Equation 28, the defining equation for $r(\kappa)$, we get:

$$\begin{aligned} r(\kappa) &= 1 + \max\left\{\left\lceil \frac{\kappa}{2} \right\rceil + \gamma + r\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1\right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + r\left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma\right)\right\} \\ &\leq 1 + \max\left\{\left\lceil \frac{\kappa}{2} \right\rceil + \gamma + 0 \ , \right. \\ &\qquad \left. \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + \left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma + (\mu + 2)\log\left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma - 2\max\{-\gamma + \mu + 1, \gamma\}\right)\right)\right\} \end{aligned}$$

$$
\begin{aligned}
&= 1 + \max\left\{\left\lceil\frac{\kappa}{2}\right\rceil + \gamma\,,\ \kappa + \mu + (\mu+2)\log\left(\left\lceil\frac{\kappa}{2}\right\rceil + \gamma - 2\max\{-\gamma+\mu+1,\gamma\}\right)\right\}\\
&\leq 1 + \max\left\{\left\lceil\frac{\kappa}{2}\right\rceil + \gamma\,,\ \kappa + \mu + (\mu+2)\log\left(\left\lceil\frac{\kappa}{2}\right\rceil - \max\{-\gamma+\mu+1,\gamma\}\right)\right\}\\
&\leq \max\left\{\left\lceil\frac{\kappa}{2}\right\rceil + \gamma + 1\,,\ \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma+\mu+1,\gamma\} + 2\right) - 1\right\}\\
&\leq \max\left\{\left\lceil\frac{\kappa}{2}\right\rceil + \gamma + 1\,,\ \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma+\mu+1,\gamma\}\right)\right\}\\
&= \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma+\mu+1,\gamma\}\right)\ .
\end{aligned}
$$

- *Case 4: κ is such that $\lfloor\kappa/2\rfloor - \gamma + \mu + 1 > \kappa_0$ and $\lceil\kappa/2\rceil + \gamma > \kappa_0$.* This case follows by similar computations as Case 2 and Case 3.

□

Lemma B.10. *The recursive function*

$$
r(\kappa) = \begin{cases} 0 & \text{if } \kappa \leq \kappa_0 \\ 2^{\lceil\kappa/2\rceil+\gamma}\cdot 2^{\lfloor\kappa/2\rfloor-\gamma+\mu+1} + 2^{\lceil\kappa/2\rceil+\gamma}\cdot r\left(\lfloor\kappa/2\rfloor - \gamma + \mu + 1\right) + 2^{\lfloor\kappa/2\rfloor-\gamma+\mu}\cdot r\left(\lceil\kappa/2\rceil + \gamma\right) & \text{if } \kappa > \kappa_0 \end{cases} \tag{29}
$$

is upper bounded by the function

$$
\widetilde{r}(\kappa) = 2^{\kappa}\kappa^{1+\max\{-\gamma+\mu+1,\gamma\}}\ .
$$

Proof. Let $\widetilde{s}(\kappa)$ be a generic function of $\kappa \in \mathbb{N}$. We derive constraints on $\widetilde{s}(\kappa)$ such that $r(\kappa) \leq 2^{\kappa}\cdot\widetilde{s}(\kappa)$. For $\kappa \in \{1,\ldots,\kappa_0\}$, the inequality $r(\kappa) \leq 2^{\kappa}\cdot\widetilde{s}(\kappa)$ follows as long as $\widetilde{s}(\kappa)$ is non-negative on $\{1,\ldots,\kappa_0\}$. For $\kappa > \kappa_0$, we distinguish between four cases:

- *Case 1: κ is such that $\lfloor\kappa/2\rfloor - \gamma + \mu + 1 \leq \kappa_0$ and $\lceil\kappa/2\rceil + \gamma \leq \kappa_0$.* By Equation 29, the defining equation for $r(\kappa)$, we get:

$$
\begin{aligned}
r(\kappa) &= 2^{\mu+1}\cdot 2^{\kappa} + 2^{\lceil\kappa/2\rceil+\gamma}\cdot r\left(\lfloor\kappa/2\rfloor - \gamma + \mu + 1\right) + 2^{\lfloor\kappa/2\rfloor-\gamma+\mu}\cdot r\left(\lceil\kappa/2\rceil + \gamma\right)\\
&\leq 2^{\mu+1}\cdot 2^{\kappa} + 2^{\lceil\kappa/2\rceil+\gamma}\cdot 0 + 2^{\lfloor\kappa/2\rfloor-\gamma+\mu+1}\cdot 0\\
&= 2^{\mu+1}\cdot 2^{\kappa}\\
&\leq 2^{\kappa}\cdot\widetilde{s}(\kappa)\ ,
\end{aligned}
$$

  where the first inequality follows by the inductive hypothesis and the last inequality is a constraint for $\widetilde{s}$.

- *Case 2: κ is such that $\lfloor\kappa/2\rfloor - \gamma + \mu + 1 > \kappa_0$ and $\lceil\kappa/2\rceil + \gamma \leq \kappa_0$.* By Equation 29, the defining equation for $r(\kappa)$, we get:

$$
\begin{aligned}
r(\kappa) &= 2^{\mu+1}\cdot 2^{\kappa} + 2^{\lceil\kappa/2\rceil+\gamma}\cdot r\left(\lfloor\kappa/2\rfloor - \gamma + \mu + 1\right) + 2^{\lfloor\kappa/2\rfloor-\gamma+\mu}\cdot r\left(\lceil\kappa/2\rceil + \gamma\right)\\
&\leq 2^{\mu+1}\cdot 2^{\kappa} + 2^{\lceil\kappa/2\rceil+\gamma}\cdot\left(2^{\lfloor\kappa/2\rfloor-\gamma+\mu+1}\cdot\widetilde{s}\left(\lfloor\kappa/2\rfloor - \gamma + \mu + 1\right)\right) + 2^{\lfloor\kappa/2\rfloor-\gamma+\mu}\cdot 0\\
&= 2^{\mu+1}\cdot 2^{\kappa} + 2^{\mu+1}\cdot 2^{\kappa}\cdot\widetilde{s}\left(\lfloor\kappa/2\rfloor - \gamma + \mu + 1\right)\\
&= 2^{\kappa}\cdot\left(2^{\mu+1} + 2^{\mu+1}\cdot\widetilde{s}\left(\lfloor\kappa/2\rfloor - \gamma + \mu + 1\right)\right)\\
&\leq 2^{\kappa}\cdot\widetilde{s}(\kappa)\ ,
\end{aligned}
$$

  where the first inequality follows by the inductive hypothesis and the last inequality is a constraint (additional to the other one that we already derived) for $\widetilde{s}$.

- *Case 3: κ is such that $\lfloor \kappa/2 \rfloor - \gamma + \mu + 1 \leq \kappa_0$ and $\lceil \kappa/2 \rceil + \gamma > \kappa_0$.* By Equation 29, the defining equation for $r(\kappa)$, we get:

$$\begin{aligned} r(\kappa) &= 2^{\mu+1} \cdot 2^{\kappa} + 2^{\lceil \kappa/2 \rceil + \gamma} \cdot r\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right) + 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu} \cdot r\left(\lceil \kappa/2 \rceil + \gamma\right) \\ &\leq 2^{\mu+1} \cdot 2^{\kappa} + 2^{\lceil \kappa/2 \rceil + \gamma} \cdot 0 + 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu} \cdot \left(2^{\lceil \kappa/2 \rceil + \gamma} \cdot \widetilde{s}(\lceil \kappa/2 \rceil + \gamma)\right) \\ &= 2^{\mu+1} \cdot 2^{\kappa} + 2^{\mu} \cdot 2^{\kappa} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor + \gamma\right) \\ &= 2^{\kappa} \cdot \left(2^{\mu+1} + 2^{\mu} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor + \gamma\right)\right) \\ &\leq 2^{\kappa} \cdot \widetilde{s}(\kappa) \ , \end{aligned}$$

  where the first inequality follows by the inductive hypothesis and the last inequality is a constraint (additional to the other two that we already derived) for $\widetilde{s}$.

- *Case 4: κ is such that $\lfloor \kappa/2 \rfloor - \gamma + \mu + 1 > \kappa_0$ and $\lceil \kappa/2 \rceil + \gamma > \kappa_0$.* By Equation 29, the defining equation for $r(\kappa)$, we get:

$$\begin{aligned} r(\kappa) &= 2^{\mu+1} \cdot 2^{\kappa} + 2^{\lceil \kappa/2 \rceil + \gamma} \cdot r\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right) + 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu} \cdot r\left(\lceil \kappa/2 \rceil + \gamma\right) \\ &\leq 2^{\mu+1} \cdot 2^{\kappa} + 2^{\lceil \kappa/2 \rceil + \gamma} \cdot \left(2^{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right)\right) \\ &\qquad + 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu} \cdot \left(2^{\lceil \kappa/2 \rceil + \gamma} \cdot \widetilde{s}(\lceil \kappa/2 \rceil + \gamma)\right) \\ &= 2^{\mu+1} \cdot 2^{\kappa} + 2^{\mu+1} \cdot 2^{\kappa} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right) + 2^{\mu} \cdot 2^{\kappa} \cdot \widetilde{s}\left(\lceil \kappa/2 \rceil + \gamma\right) \\ &= 2^{\kappa} \cdot \left(2^{\mu+1} + 2^{\mu+1} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right) + 2^{\mu} \cdot \widetilde{s}\left(\lceil \kappa/2 \rceil + \gamma\right)\right) \\ &\leq 2^{\kappa} \cdot \widetilde{s}(\kappa) \ , \end{aligned}$$

  where the first inequality follows by the inductive hypothesis and the last inequality is a constraint (additional to the other three that we already derived) for $\widetilde{s}$.

Overall, we require that

$$\widetilde{s}(\kappa) \geq \begin{cases} 0 & \text{if } \kappa \leq \kappa_0 \\ 2^{\mu+1} & \text{if Case 1} \\ 2^{\mu+1} + 2^{\mu+1} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right) & \text{if Case 2} \\ 2^{\mu+1} + 2^{\mu} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor + \gamma\right) & \text{if Case 3} \\ 2^{\mu+1} + 2^{\mu+1} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right) + 2^{\mu} \cdot \widetilde{s}\left(\lceil \kappa/2 \rceil + \gamma\right) & \text{if Case 4} \end{cases} .$$

By inspection, $\widetilde{s}(\kappa) \stackrel{\text{def}}{=} \kappa^{1+\max\{-\gamma+\mu+1,\gamma\}}$ works for all $\kappa \in \mathbb{N}$. □

References

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: efficient verification via secure computation. In *ICALP '10: Proceedings of the 37th International Colloquium on Automata, Languages, and Programming*, pages 152–163, Berlin, Heidelberg, 2010. Springer-Verlag.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in FOCS '92.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. Preliminary version in FOCS '92.

[BCCT11] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. Cryptology ePrint Archive, Report 2011/443, 2011.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Composition and bootstrapping for SNARKs and proof-carrying data. Cryptology ePrint Archive, 2012.

[BF90] Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '90, pages 37–48. 1990.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC '91: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 21–32, New York, NY, USA, 1991. ACM.

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 390–420, London, UK, 1993. Springer-Verlag.

[BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *CCC '02: Proceedings of the 17th IEEE Annual Conference on Computational Complexity*, pages 194–203, Washington, DC, USA, 2002. IEEE Computer Society. We reference the version available online at http://www.wisdom.weizmann.ac.il/~oded/PS/ua-rev3.ps.

[Bha05] Arnab Bhattacharyya. Implementing probabilistically checkable proofs of proximity. Technical Report MIT-CSAIL-TR-2005-051, MIT, 2005. Available at http://dspace.mit.edu/handle/1721.1/30562.

[BSCGT12] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems, 2012. Cryptology ePrint Archive.

[BSGH+04] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs and applications to coding. In *STOC '04: Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 1–10, New York, NY, USA, 2004. ACM. Full version available at http://people.seas.harvard.edu/~salil/research/shortPCP.pdf.

[BSGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Short PCPs verifiable in polylogarithmic time. In *CCC '05: Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pages 120–134, Washington, DC, USA, 2005. IEEE Computer Society.

[BSGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006. Preliminary versions of this paper have appeared in Proceedings of the 36th ACM Symposium on Theory of Computing and in Electronic Colloquium on Computational Complexity.

[BSHR05] Eli Ben-Sasson, Prahladh Harsha, and Sofya Raskhodnikova. Some 3CNF properties are hard to test. *SIAM Journal on Computing*, 35(1):1–21, 2005.

[BSS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM Journal on Computing*, 38(2):551–607, 2008. Preliminary version appeared in STOC '05.

[BSSVW03] Eli Ben-Sasson, Madhu Sudan, Salil Vadhan, and Avi Wigderson. Randomness-efficient low degree tests and short pcps via epsilon-biased sets. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, STOC '03, pages 612–621, 2003.

[CKV10] Kai-Min Chung, Yael Kalai, and Salil Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO '10: Proceedings of the 30th Annual International Cryptology Conference on Advances in Cryptology*, pages 483–501, Berlin, Heidelberg, 2010. Springer-Verlag. Full version online at http://eprint.iacr.org/2010/241.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS '10: Proceedings of the 1st Symposium on Innovations in Computer Science*, pages 310–331, Beijing, China, 2010. Tsinghua University Press.

[DCL08] Giovanni Di Crescenzo and Helger Lipmaa. Succinct NP proofs from an extractability assumption. In *CiE '08: Logic and Theory of Algorithms, 4th Conference on Computability in Europe*, pages 175–185. Springer, 2008.

[DFH11] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. Cryptology ePrint Archive, Report 2011/508, 2011.

[Din07] Irit Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3):12, 2007.

[DR04] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the pcp-theorem. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 155–164, Washington, DC, USA, 2004. IEEE Computer Society.

[FGL+96] Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996. Preliminary version in FOCS '91.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 169–178, 2009.

[Gen11] Craig Gentry. Fully homomorphic encryption without bootstrapping, 2011. Cryptology ePrint Archive, Report 2011/277.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO '10: Proceedings of the 30th Annual International Cryptology Conference on Advances in Cryptology*, pages 465–482, Berlin, Heidelberg, 2010. Springer-Verlag. http://eprint.iacr.org/2009/547.

[GH11] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *EUROCRYPT '11: Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2011. Full version as Cryptology ePrint Archive, Report 2010/520.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier CS-proofs. Cryptology ePrint Archive, Report 2011/456, 2011.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '10, pages 321–340, 2010.

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Information Processing Letters*, 43(4):169–174, 1992.

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and pcps of almost-linear length. *Journal of the ACM*, 53:558–655, July 2006. Preliminary version in STOC '02.

[HS00] Prahladh Harsha and Madhu Sudan. Small PCPs with low query complexity. *Computational Complexity*, 9(3–4):157–201, Dec 2000. Preliminary version in STACS '91.

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *CCC '07: Proceedings of the Twenty-Second Annual IEEE Conference on Computational Complexity*, pages 278–291, Washington, DC, USA, 2007. IEEE Computer Society.

[IO11] Yuval Ishai and Rafail Ostrovsky. Linear interactive proofs and the complexity of verification, 2011. Unpublished manuscript.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *STOC '92: Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 723–732, New York, NY, USA, 1992. ACM.

[Lip90] Richard J. Lipton. Efficient checking of computations. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '90, pages 207–215, 1990.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 9th Theory of Cryptography Conference on Theory of Cryptography*, TCC '12, pages 169–189, 2012.

[LN97] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Cambridge University Press, Cambridge, UK, second edition edition, 1997.

[Mat08] Todd Mateer. *Fast Fourier Transform algorithms with applications*. PhD thesis, Clemson University, 2008.

[Mei09] Or Meir. Combinatorial PCPs with efficient verifiers. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 463–471. IEEE Computer Society, 2009.

[Mei12] Or Meir. Combinatorial pcps with short proofs. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity*, CCC '12, 2012.

[Mic98] Silvio Micali. Computationally-sound checkers. In *MFCS '98: Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science*, pages 94–116, London, UK, 1998. Springer-Verlag.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000. Preliminary version appeared in FOCS '94.

[Mie09] Thilo Mie. Short PCPPs verifiable in polylogarithmic time with o(1) queries. *Annals of Mathematics and Artificial Intelligence*, 56:313–338, 2009.

[MR08] Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *Journal of the ACM*, 57:1–29, June 2008. Preliminary version appeared in FOCS '08.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, USA, 1994.

[PS94] Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *STOC '94: Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 194–203, New York, NY, USA, 1994. ACM.

[RS97] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, STOC '97, pages 475–484, 1997.

[RV09] Guy N. Rothblum and Salil Vadhan. Are PCPs inherent in efficient arguments? In *CCC '09: Proceedings of the 24th IEEE Annual Conference on Computational Complexity*, pages 81–92, Washington, DC, USA, 2009. IEEE Computer Society.

[Spi95] Daniel Spielman. *Computationally Efficient Error-Correcting Codes and Holographic Proofs*. PhD thesis, MIT, Mathamatics Department, May 1995.

[Sze05] Mario Szegedy. Probabilistic verification and non-approximability. In *Handbook of Combinatorial Optimization*, pages 83–191. Springer US, 2005.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC '08: Proceedings of the 5th Theory of Cryptography Conference on Theory of Cryptography*, pages 1–18, Berlin, Heidelberg, 2008. Springer-Verlag.

[vzGG03] Joachim von zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[WB86] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction of algebraic block codes, December 1986.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de