Electronic Colloquium on Computational Complexity, Revision 3 of Report No. 45 (2012)

On the Concrete Efficiency of Probabilistically-Checkable Proofs*

Eli Ben-Sasson† eli@cs.technion.ac.il Technion

Alessandro Chiesa† alexch@csail.mit.edu MIT

Daniel Genkin† danielg3@cs.technion.ac.il Technion

Eran Tromer‡ tromer@cs.tau.ac.il Tel Aviv University

December 28, 2012

Abstract

Probabilistically-Checkable Proofs (PCPs) form the algorithmic core that enables fast verification of long computations in many cryptographic constructions. Yet, despite the wonderful asymptotic savings they bring, PCPs are also the infamous computational bottleneck preventing these powerful cryptographic constructions from being used in practice. To address this problem, we present several results about the computational efficiency of PCPs.

We construct the first PCP where the prover and verifier time complexities are *quasi-optimal* (i.e., optimal up to polylogarithmic factors). The prover and verifier are also *higly-parallelizable*, and these computational guarantees hold even when proving and verifying the correctness of random-access machine computations. Our construction is explicit and has the requisite properties for being used in the cryptographic applications mentioned above.

Next, to better understand the efficiency of our PCP, we propose a new efficiency measure for PCPs (and their major components, 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 is cheaper than performing naive verification (i.e., rerunning the computation); our definition accounts for *both* the prover and verifier complexity.

We then show that our PCP has a *finite* concrete-efficiency threshold. That such a PCP exists does not follow from existing works on PCPs with polylogarithmic-time verifiers.

As in [Ben-Sasson and Sudan, STOC '05], PCPs of proximity for Reed–Solomon (RS) codes are the main component of our PCP. We construct a PCP of proximity that reduces the concrete-efficiency threshold for testing proximity to RS codes from 2^{683} in their work to 2^{43}, which is tantalizingly close to practicality.

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 **4**
1.1 Motivation . . . 4
1.2 Asymptotic Time Efficiency . . . 5
1.3 Defining Concrete Efficiency . . . 5
1.4 A PCP System with $B < \infty$ for Every Polynomial Cost Function . . . 7
1.5 Concrete Efficiency of PCPs of Proximity for Reed–Solomon Codes . . . 8
1.6 Properties for Cryptographic Constructions . . . 10
1.7 Algorithmic Specifications . . . 10
1.8 The Road Onwards . . . 11

2 Main Results **11**
2.1 Theorem 1: Quasi-Optimal PCPs For RAMs . . . 11
2.2 Defining the Concrete-Efficiency Threshold of a PCP System . . . 11
2.3 Theorem 2: A B-efficient PCP for sCSAT with $B < \infty$ . . . 13
2.4 Theorem 3: Concrete Efficiency of Proximity Testing to Reed–Solomon . . . 14
2.5 Theorem 4: Properties for Cryptographic Constructions . . . 15

3 Open Questions **16**

4 Definitions **18**
4.1 Reed–Solomon and Reed–Muller Codes . . . 18
4.2 Notions of Distance . . . 18
4.3 PCPs and PCPPs . . . 19

5 Proof of Theorem 1 **21**

6 Proof of Theorem 2 **24**

7 A Succinct Algebraic Constraint Satisfaction Problem **25**

8 A PCP for sACSP **28**
8.1 The Construction At High Level . . . 28
8.2 The Construction In Detail . . . 29

9 Proof of Theorem 3 **39**
9.1 Step 1: Improving the Universal Constant of Bivariate Testing . . . 39
9.2 Step 2: Improving the Soundness Analysis of the Recursive Construction . . . 40
9.3 Step 3: Putting Things Together . . . 44

10 Improved Universal Constant of Bivariate Testing **46**
10.1 Some Basic Lemmas . . . 46
10.2 The PS Bivariate Testing Theorem . . . 46
10.3 The Universal Constant for Bivariate Testing . . . 48
10.4 Putting Things Together . . . 50

11 Improved Soundness for V_{RS} and V_{VRS} **51**
11.1 Soundness Analysis of $V_{\mathrm{RS},=}$ . . . 52
11.2 Soundness Analysis of $V_{\mathrm{RS},<}$ . . . 66
11.3 Soundness Analysis of $V_{\mathrm{RS},>}$ . . . 67

11.4 Soundness Analysis of V_{RS} . . . 68
11.5 Soundness Analysis of V_{VRS} . . . 68

12 Proof of Theorem 4 **70**
12.1 Universal Arguments and Our Relaxation . . . 71
12.2 Relatively-Efficient Oracle Construction . . . 74
12.3 Non-Adaptive Verifier . . . 75
12.4 Efficient Reverse Sampling . . . 89
12.5 Proof Of Knowledge . . . 98
12.6 Obtaining (Almost) Universal Arguments . . . 100

A Work On Fast Verification Of Long Computations **103**

B PCP Algorithmic Specifications **105**
B.1 PCP Prover Specifications . . . 107
B.2 PCP Verifier Specifications . . . 114
B.3 Finite Field Algorithms . . . 123

C Complexity Analysis **131**
C.1 Complexity Analysis of $P_{\text{RS},=}$ and $V_{\text{RS},=}$ . . . 131
C.2 Complexity Analysis of $P_{\text{RS},<}$ and $V_{\text{RS},<}$ . . . 132
C.3 Complexity Analysis of $P_{\text{RS},>}$ and $V_{\text{RS},>}$ . . . 132
C.4 Complexity Analysis of P_{RS} and V_{RS} . . . 132
C.5 Complexity Analysis of P_{VRS} and V_{VRS} . . . 133
C.6 Complexity Analysis of V_{aRS} and V_{aVRS} . . . 133
C.7 Complexity Analysis of P_{sACSP} and V_{sACSP} . . . 134
C.8 Solving Recursions . . . 134

References **138**

1 Introduction

The study of *probabilistically-checkable proofs* (PCPs) [AS98, ALM$^+$98] was initiated by Babai et al. [BFLS91] and Feige et al. [FGL$^+$96] with two very different motivations. Babai et al. focused on "positive" applications of PCPs: enabling fast verification of long computations. Instead, Feige et al. focused on "negative" applications of PCPs: proving inapproximability results. In this paper, we study properties of PCPs that are relevant to positive applications.

1.1 Motivation

Given a program $\mathbb{P}$ (say, written in `C++` or some other random-access memory model), input $\mathtt{x}$, and time bound T, a verifier wishes to check whether a witness $\mathtt{w}$ makes $\mathbb{P}(\mathtt{x}, \mathtt{w})$ accept within T time steps. Without the help of a prover and only having access to $\mathtt{w}$, the verifier can only perform the naive verification procedure: run $\mathbb{P}(\mathtt{x}, \mathtt{w})$ for at most T steps and see if it halts and accepts.

As [BFLS91] showed, a PCP saves the verifier the cost of performing this naive verification procedure, and enables instead a faster probabilistic verification procedure: a PCP verifier can run in only $\mathsf{poly}(|\mathbb{P}| + |\mathtt{x}| + \log T)$ time when having oracle access to a PCP proof string π of length $\mathsf{poly}(|\mathbb{P}| + |\mathtt{x}| + T)$ generated by a PCP prover (who is given $\mathtt{w}$ as auxiliary input). More recent works [BSS08, BSGH$^+$05, Mie09] achieved proof length $T \cdot \mathsf{poly}(|\mathbb{P}| + |\mathtt{x}| + \log T)$. As T grows larger (and viewing $|\mathbb{P}|$ and $|\mathtt{x}|$ as relatively small) the saving in running time becomes more dramatic, and at some point running the PCP verifier is much cheaper than performing naive verification.

Due to these properties, PCPs have been used as the "verification engine" in many cryptographic protocols that enable various flavors of fast verification of long computations, including *succinct interactive arguments* [Kil92, Mic00, BG08], succinct *non-interactive* arguments (SNARGs) [DCL08, BCCT12a, DFH12, GLR11], and *proof-carrying data* [CT10, CT12].

Unfortunately, PCPs are also the notorious computational bottleneck of all these cryptographic protocols: despite the great practical potential of fast verification of long computations, any construction relying on PCPs is dismissed as impractical.[1] Indeed, known PCP constructions with $\mathsf{poly}(|\mathbb{P}| + |\mathtt{x}| + \log T)$-time verifiers have huge overheads, indicating that real savings are made only for prohibitively large T. Moreover, 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 generated via efficient algorithms — can be much larger, so much so as to kill any prospect of using a PCP system in practice.

Therefore, two crucial parameters of interest from the point of view of using PCPs for fast verification of long computations are the *prover running time* and the *verifier running time*.[2] While 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 (informal) question that motivates our work. A *galactic algorithm* is an algorithm with great asymptotics that, however, is never used to actually

[1] See Appendix A for a summary of flavors of solutions to the problem of fast verification of long computations.

[2] 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. (We are not interested in randomness-efficient repetition, such as that of [BSS08, Proposition 2.9].)

compute anything in practice [Lip10]. Are all PCPs galactic algorithms or are there PCPs with "terrestrial" efficiency?

In this paper we present several promising results on the computational efficiency of PCPs, both in the asymptotic and concrete senses. We discuss these next.

1.2 Asymptotic Time Efficiency

Previous PCP constructions with polylogarithmic-time verifiers [BFLS91, BSGH$^+$05, Mie09, Mei09] did not obtain quasilinear-time provers.

Our first main theorem is obtaining the first PCP system where the time complexity of both the prover and verifier is *quasi-optimal* (i.e., optimal up to polylogarithmic factors).

This PCP system has two additional desirable features. First, both the prover and verifier are *highly parallelizable*; especially in the prover, low parallel complexity is very desirable, because it corresponds to the "lag time" for generating the PCP proof, relative to the completion time of the original computation. Second, these computational guarantees hold even when proving and verifying the correctness of *programs* (more precisely, random-access machines).

Overall, our result thus establishes that, from an asymptotic viewpoint, the time complexity of PCPs is essentially as good as one might hope for.

Theorem 1 (informal)**.** *There is a PCP system where, to prove and verify that a program* $\mathbb{P}$ *accepts the input* $(\mathtt{x}, \mathtt{w})$ *within* T *steps for some witness* $\mathtt{w}$ *(with* $|\mathtt{x}|, |\mathtt{w}| \leq T$*),*

Sequential Time:

- *given* $(\mathbb{P}, \mathtt{x}, T, \mathtt{w})$*, the prover runs in time* $(|\mathbb{P}| + T) \cdot \mathsf{polylog}(T)$*;*
- *given* $(\mathbb{P}, \mathtt{x}, T)$*, the verifier runs in time* $(|\mathbb{P}| + |\mathtt{x}|) \cdot \mathsf{polylog}(T)$*.*

Parallel Time:

- *the prover runs in parallel time* $O((\log T)^2)$*, when also given as input the transcript of computation of* $\mathbb{P}$ *on* $(\mathtt{x}, \mathtt{w})$*;*
- *the verifier runs in parallel time* $O((\log T)^2)$*.*

The construction of our PCP system relies and builds on the quasilinear-size Reed–Solomon proximity proofs of [BSS08]. As discussed, our contribution lies in obtaining the aforementioned efficiency properties in the prover and verifier algorithms. In order to obtain them, we show how to suitably leverage computational properties of linearized polynomials [LN97, Section 2.5] in finite field extensions of $\mathbb{F}_2$ [BSGH$^+$05] and additive-FFT methods [Mat08].

The high-level proof strategy for the theorem is as follows. First, following and generalizing [BSGH$^+$05], we introduce a family of NEXP-complete *succinct algebraic constraint satisfaction problems* (sACSPs); our definition of an sACSP attempts to *simultaneously* capture essential ingredients that make the problem amenable to probabilistic checking as well as general enough to allow for efficient reductions from less "structured" problems. In order to (efficiently) reduce correctness of program computations to satisfiability of sACSPs, we rely on the reductions of Ben-Sasson et al. [BSCGT13]. Second, we construct a PCP system, with the required efficiency, to probabilistically check satisfiability of sACSP problems.

1.3 Defining Concrete Efficiency

An asymptotic understanding of PCPs, however, is insufficient for investigating the practical feasibility of PCPs. For comparison, in coding theory, extensive research effort has been invested in optimizing codes for a given block length, rate, decoding radius, and so on [Ber68, HB98]. Similar questions arise for PCPs. We thus initiate the study of the *concrete* (as opposed to *asymptotic*) efficiency of PCP constructions, from a perspective that is informed by the efficiency concerns arising in positive applications of PCPs. In this direction, we present three contributions: the first is a definitional one, the remaining two are of a mathematical nature.

We begin by defining the *concrete-efficiency threshold* B of a given PCP system. It is the answer to the following natural question.

> **Question:** What is the smallest T for which using the PCP system to verify computations of length at least T is cheaper than naive verification?

To concretize our question, we need to agree on (i) the cost of using a PCP system and (ii) the cost of naive verification.

(i) Cost of using a PCP system. A PCP system consists of *two* algorithms: a PCP prover P and a PCP verifier V. As discussed, two important quantities are the prover running time t_P and the verifier running time t_V. Because it is often possible to reduce one at the expense of the other, a useful cost measure should simultaneously take into account both running times. We therefore seek a bivariate *cost function* $\mathbf{C}$ capturing the combined cost.[3] However, t_P and t_V are not "on the same scale": we expect t_P to be always much larger than t_V. To place them on the same scale so that they can be combined "fairly", we normalize them into unitless *multiplicative overhead* numbers, and give these as input to the cost function $\mathbf{C}$.

Specifically, in an ideal world where the verifier trusts the prover, the prover runs the computation to deduce what is the answer bit (i.e., accept or not), and then communicates this bit to the verifier; let i_P and i_V denote the running times of the prover and verifier in this ideal world. Then i_P and i_V are the "natural units" of the prover and verifier respectively. We thus give as input to $\mathbf{C}$ the two (unitless) ratios $\mathsf{t}_P/\mathsf{i}_P$ and $\mathsf{t}_V/\mathsf{i}_V$.

(ii) Cost of naive verification. We reason similarly on how to compute the cost of naive verification, i.e., the cost of verification without the use of PCPs. In the ideal world, the verifier runs in time i_V as before. In contrast, in a real world where the PCP system is not used (and the verifier does not trust anyone but himself), the verifier is forced to conduct the computation himself to figure out the answer bit; this takes time i_P (i.e., the same time as that needed by the prover in the ideal world). Thus, the natural unit of naive verification is i_V, and its unitless cost is thus $\mathsf{i}_P/\mathsf{i}_V$.

Concrete-efficiency threshold. Given (i) and (ii) above, we define the concrete-efficiency threshold of a PCP system as the smallest problem size beyond which the (combined) cost $\mathbf{C}(\mathsf{t}_P/\mathsf{i}_P, \mathsf{t}_V/\mathsf{i}_V)$ is always smaller than the cost $\mathsf{i}_P/\mathsf{i}_V$. We summarize this via the following informal definition:

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

$$\mathbf{C}\begin{pmatrix}\text{PCP proving} & , & \text{PCP verification} \\ \text{overhead for } y & & \text{overhead for } y\end{pmatrix} < \begin{pmatrix}\text{naive verification} \\ \text{overhead for } y\end{pmatrix} .$$

If no such integer exists, $B := \infty$.

[3]Of course, prover and verifier running time are not the *only* important quantities. For instance, the parallel complexity of the prover is also important, because it corresponds to the "lag time" of producing a PCP proof; hence, it is also meaningful to let $\mathbf{C}$ take this quantity as a third input. Similarly other quantities of potential interest, such as the space complexity of the prover. For simplicity, we study the most basic case where $\mathbf{C}$ takes prover and verifier running time.

Depending on the application, different choices of cost function $\mathbf{C}$ may be meaningful.[4] In this paper, we focus on *polynomial cost functions*, i.e., $\mathbf{C}(a, b)$ is a bivariate polynomial over the reals (that is at least linear in both variables). We believe that polynomial cost functions are rich enough to model many scenarios.

A model for concrete efficiency. In order formalize Definition 1, we still need to choose (a) a computational model (through which PCP provers and verifiers are represented) as well as (b) a "reference" complete problem relative to which a given PCP is constructed. However, having moved from an asymptotic study of efficiency to a concrete one, we must be careful about how we make these choices.

The issue of picking the "right" computational model does not arise in typical algorithmic analysis because many algorithms are naturally associated with certain complexity measures that are meaningful in both the asymptotic and concrete settings. E.g., matrix-multiplication algorithms are naturally studied via the number of arithmetic operations they require, graph algorithms are studied via number of vertex/edge queries, etc., and these complexity measures are quite meaningful in concrete settings, for example, to answer questions analogous to ours: "What is the smallest matrix size for which Strassen's algorithm costs less than naive cubic-time algorithm?".

Returning to a PCP system, the natural objects on which it operates are programs (cf. Theorem 1) specified to run on some machine. In the asymptotic setting it does not matter which machine is used to execute them, provided it is universal. But moving to the concrete world, fixing an *arbitrary* universal machine as a model computation would make the definition of B meaningless. Indeed, if a PCP system has $B < \infty$ when computing on a universal machine U, then there is also a universal machine U' that has all problems of size less than B precomputed (and for larger sizes U' simulates U), and thus the same PCP system has $B' = 0$ when choosing U' as a computational model. This can be avoided by specifying a "natural" universal machine *explicitly*; however, this would result in a cumbersome and somewhat arbitrary definition.

We therefore suggest to study the concrete efficiency of PCP systems by using *circuits*. In particular, in our Definition 1, we assume that (a) provers and verifiers are specified as Boolean circuits, and (b) the PCP system is constructed for a NEXP-complete language, Succinct Circuit Sat (or sCSAT for short), which is the language of satisfiable circuits that are *succinctly* represented.[5] Indeed, circuits appear to be a natural and meaningful computational model for studying concrete efficiency, especially because we can be explicit about the exact model of computation without much work (e.g., by specifying a natural Boolean basis and settings for fan-in and fan-out). See Section 2.2 for details.

Henceforth, our goal is to construct B-efficient PCP systems for sCSAT, with as small a B as possible, for various polynomial cost functions $\mathbf{C}$.

1.4 A PCP System with $B < \infty$ for Every Polynomial Cost Function

Our first result about the concrete efficiency of PCPs is constructing a PCP system for sCSAT that has *finite* concrete-efficiency threshold for *every* polynomial cost function $\mathbf{C}$. Indeed, Theorem 1 implies a PCP system for sCSAT with analogous quasi-optimal asymptotic efficiency

[4] E.g., if one is interested in studying the threshold corresponding to when the combined cost becomes less than *half* (rather than merely less than) the cost of naive verification, one can use the cost function $\mathbf{C}'(a, b) := 2{\cdot}\mathbf{C}(a, b)$.

[5] 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!

(where running time is replaced by circuit size and parallel time by circuit depth), from which we can deduce:

Theorem 2 (informal). *There exists a PCP system for* sCSAT *with concrete-efficiency threshold* $B(\mathbf{C}) < \infty$ *for every polynomial cost function* $\mathbf{C}$.

Note that it is not a-priori clear that PCP systems with a finite efficiency threshold (even for one polynomial cost function) exist. 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 T, and therefore have an infinite efficiency threshold (i.e., none exists) [FGL+96, AS98, ALM+98, PS94, RS97, HS00, BSSVW03, BSGH+04, GS06, Din07, BSS08, MR08, Mei12]. Turning to PCP constructions with polylogarithmic-time verifiers [BFLS91, BSGH+05, Mie09, Mei09], these focus on minimizing the verifier's running time, and only show that the prover's running time is bounded by a polynomial in T, which, once again, is not enough to obtain a finite concrete-efficiency threshold. Another important issue is the dependence of the running time of the prover or verifier on the statement size: if it is too large of a polynomial, then again the concrete-efficiency threshold is ∞.

Having established via our Theorem 2 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 2 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, entails 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.

1.5 Concrete Efficiency of PCPs of Proximity for Reed–Solomon Codes

Our PCP construction from the proof of Theorem 1 is an *algebraic PCP* whose technical heart and "heaviest" component, like other algebraic PCPs, is a *low-degree test*: 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)* for a code ensemble.

As an intermediate step to computing the concrete-efficiency threshold of our PCP system (which, as discussed, is quite challenging), we study the concrete-efficiency threshold of PCPPs for Reed–Solomon codes over fields of characteristic 2; these, as in [BSS08, BSGH+05], lie at the core of our PCP system. We thus 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 proving and the verification overheads, we compute a combined cost based on the *rate* of the codeword and its PCPP proof, 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** *with respect to cost function* $\mathbf{C}$ *if* $\hat{B}$ *is the smallest integer such that for all* $k \geq \hat{B}$,

$$\mathbf{C}\left(\frac{n(k) + L(k)}{k}, Q(k)\right) < k$$

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.

Definition 2 is an "information-theoretic" version of Definition 1 as it disregards the *computational* cost of the PCPP prover for producing the proximity proof and of the PCPP verifier for generating queries and examining answers, and focuses instead on information-theoretic measures (see Remark 2.8). In our PCPP construction (and, ultimately, also in our PCP construction), through the use of additive-FFT techniques, we ensure that the computational costs only account for small additional logarithmic factors relative to the PCPP proof length and query complexity; we thus believe that in our case it is meaningful to study the (cleaner) information-theoretic efficiency measure of Definition 2, as a proxy for its computational equivalent, for the PCPPs we construct.

As in [BSS08, BSGH^{+}05], the relevant code ensemble for our PCP construction from the proof of Theorem 2 is the ensemble $\mathcal{E}_{\mathrm{RS}}$ of Reed–Solomon (RS) codes evaluated over linear subspaces of fields of characteristic 2.[6] Let us denote $\hat{B}_{\mathrm{RS}}(\mathbf{C})$ the concrete-efficiency threshold of a given PCPP system for $\mathcal{E}_{\mathrm{RS}}$ (relative to cost function $\mathbf{C}$).

Our third main result is a PCPP system for $\mathcal{E}_{\mathrm{RS}}$ with an analysis that yields much smaller $\hat{B}_{\mathrm{RS}}(\mathbf{C})$ than values implied by previous work; in fact, we can *explicitly* derive an upper bound for the $\hat{B}_{\mathrm{RS}}(\mathbf{C})$ of our PCPP construction, for *any* (efficiently-computable) cost function $\mathbf{C}$.[7] Of course, the value of $\hat{B}_{\mathrm{RS}}(\mathbf{C})$ changes from cost function to cost function. Thus, for concreteness, we choose to state our theorem relative to the natural polynomial cost function $\mathbf{C}_{\times}(a,b) := a \cdot b$.[8] In this case, the analysis of the PCPP system for Reed–Solomon in [BSS08] only shows an efficiency threshold of $\hat{B}_{\mathrm{RS}}(\mathbf{C}_{\times}) \leq 2^{683}$, which is an astronomical upper bound.[9] Our third main theorem significantly improves on it and obtains a PCPP system with small(er) $\hat{B}_{\mathrm{RS}}(\mathbf{C}_{\times})$:

Theorem 3 (informal)**.** *There exists a PCPP system $\mathcal{E}_{\mathrm{RS}}$ with $\hat{B}_{\mathrm{RS}}(\mathbf{C}_{\times}) \leq 2^{43}$.*

Problem sizes on the order of 2^{43} 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.

We emphasize that Theorem 3 does not follow from "experiments" or from exploiting "loopholes" of the new definition we have introduced. Instead, it originates from a much more refined understanding of the soundness of testing proximity to RS codes; the soundness analysis is the main technical challenge in this setting, and is a theoretical question that requires appropriate proofs to address. Specifically, our proof strategy for proving Theorem 3 is in two steps: (a) improve the Polishchuk–Spielman Bivariate Testing Theorem [PS94]; and (b) develop a class of constructions that generalize the RS 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 problem size and construction from the class. Armed with this understanding, we can use Definition 2 to concisely express our improvement over the status quo.

[6]More precisely, for computational reasons, we shall work in the slightly more general setting of *affine* subspaces.

[7]See `http://people.csail.mit.edu/alexch/research/tppcp/` for a Mathematica notebook that, given any (efficiently-computable) cost function $\mathbf{C}$, computes $\hat{B}_{\mathrm{RS}}(\mathbf{C})$ for our construction and (as a comparison) for the construction in [BSS08].

[8]The other natural polynomial cost function is $\mathbf{C}_{+}(a,b) := a+b$. Since $\mathbf{C}_{+}(a,b) \leq \mathbf{C}_{\times}(a,b)$, we choose $\mathbf{C}_{\times}$ because it is a tougher requirement to meet.

[9]In a preprint version of this paper, we claimed threshold computations that were smaller for both [BSS08] and our construction, since we computed the threshold by dividing by the block length $n(k)$ instead of the dimension k. This has been corrected.

1.6 Properties for Cryptographic Constructions

We further show that our PCP construction from the proof of Theorem 1 has the requisite properties for use in cryptographic constructions. Indeed, a PCP system itself 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 itself to satisfy additional properties.

Perhaps the most important positive application of PCPs is the construction of *succinct arguments* [Kil92, Mic00, BG08]. In a succinct argument, a verifier again wishes to check whether a witness $\mathtt{w}$ causes $\mathbb{P}(\mathtt{x}, \mathtt{w})$ to accept within T time steps, for a given program $\mathbb{P}$, input $\mathtt{x}$, and time bound T. By interacting with a computationally-bounded prover running in time $\mathsf{poly}(\kappa + |\mathbb{P}| + |\mathtt{x}| + T)$, where κ denotes the security parameter, the verifier can check such a statement by running in time merely $\mathsf{poly}(\kappa + |\mathbb{P}| + |\mathtt{x}| + \log T)$.

One additional property that is often essential in constructions of succinct arguments is *proof of knowledge*, which makes it possible to delegate "cryptographic computations" [BCCT12a] and to recursively compose non-interactive proofs [Val08, CT10, BSW12, BCCT12b, CT12]. It is used in constructions of computationally-sound proofs of knowledge in the random-oracle model [Mic00, Val08], SNARKs [BCCT12a], and proof-carrying data [CT10, CT12].

Some applications require even more properties. For example, the construction of *universal arguments* by Barak and Goldreich [BG08] requires, beyond proof of knowledge, also a *non-adaptive verifier* and the existence of an *efficient reverse sampler* for the PCP queries.

Focusing on such positive applications of PCPs, our third theorem states that our PCPs indeed have the requisite additional properties.

Theorem 4 (informal)**.** *The PCPs from Theorem 1 satisfy the requisite properties for the construction of a weaker (yet still useful) variant of universal arguments [BG08]. In particular, our PCPs also satisfy the requisite properties for the construction of computationally-sound proofs of knowledge [Mic00, Val08], SNARKs [BCCT12a], and proof-carrying data [CT10, CT12].*

Furthermore, due to the quasi-optimal running times of the PCP prover and verifier from Theorem 1, we obtain the first constructions of the aforementioned cryptographic primitives in which the prover and verifier have quasi-optimal time complexity.

1.7 Algorithmic Specifications

Beyond the mathematical results, we offer an additional technical contribution: we include a *complete and detailed* algorithmic specification of every non-elementary algorithm used by the verifier *and* the prover in our PCP construction. This provides a reference for others wishing to study the practical efficiency of PCPs.

The specification makes it clear how the various components of our construction come together, and how we leverage the computational properties of linearized polynomials [LN97, Section 2.5] in finite field extensions of $\mathbb{F}_2$ and additive-FFT methods [Mat08] for ensuring the efficiency properties of our construction. We also provide a detailed complexity analysis of the proof length, randomness, and query complexity. For more details see Appendix B and Appendix C.

1.8 The Road Onwards

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. For instance, we believe that the notion of a concrete-efficiency threshold 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.

2 Main Results

In the previous section we discussed our main results at high level. We now formally state these; along the way, we shall give pointers to the relevant technical sections, which contain their proofs. We begin by stating our first main result about a PCP system for RAM computations (Section 2.1); we then give the definition of the concrete-efficiency threshold of a PCP system (Section 2.2); we then state our second main result about the existence of a PCP system with finite concrete-efficiency threshold (Section 2.3); we then define the concrete-efficiency threshold for PCPPs and state our third main result about PCPPs for Reed–Solomon codes (Section 2.4); finally, we discuss cryptographic properties of the PCPs we construct (Section 2.5).

2.1 Theorem 1: Quasi-Optimal PCPs For RAMs

In Section 1.2 we informally stated our first main result. We now give a formal statement of the theorem; its proof is in Section 5.

Define $\mathrm{BH}_{\mathsf{RAM}}$ to be the language consisting of all triples $(M, \mathtt{x}, T)$, where M is a random-access machine [CR72, AV77], $\mathtt{x}$ is an input, and T is a time bound (with $|\mathtt{x}| \leq T$), for which there exists a witness $\mathtt{w}$ (with $|\mathtt{w}| \leq T$) such that M accepts $(\mathtt{x}, \mathtt{w})$ after at most T time steps.[10]

Theorem 1 (restated)**.** *There exists a PCP system* $(P_{\mathrm{RAM}}, V_{\mathrm{RAM}})$ *for* $\mathrm{BH}_{\mathsf{RAM}}$ *where: for every* $(M, \mathtt{x}, T) \in \mathrm{BH}_{\mathsf{RAM}}$ *with a witness* $\mathtt{w}$*,*

- $P_{\mathrm{RAM}}(M, \mathtt{x}, T, \mathtt{w})$ *runs in sequential time* $(|M|+T)\cdot\mathsf{polylog}(T)$ *and parallel time* $O((\log T)^2)$ *when given the transcript of computation of* M *on* $(\mathtt{x}, \mathtt{w})$*.*
- $V_{\mathrm{RAM}}(M, \mathtt{x}, T)$ *runs in sequential time* $(|M|+|\mathtt{x}|)\cdot\mathsf{polylog}(T)$ *and parallel time* $O((\log T)^2)$*.*

2.2 Defining the Concrete-Efficiency Threshold of a PCP System

In Section 1.3 we informally introduced the concrete-efficiency threshold of a PCP system (see Definition 1). We now give a formal definition for it.

As discussed, 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

[10] While the witness $\mathtt{w}$ for an instance $y = (M, \mathtt{x}, t)$ has size at most t, there is *no a-priori* polynomial bounding t in terms of $|\mathtt{x}|$. Also, the restriction that $|y|, |\mathtt{w}| \leq t$ simplifies notation but comes with essentially no loss of generality: see [BSCGT13] for a discussion of how to deal with "large inputs" (i.e., $\mathtt{x}$ or $\mathtt{w}$ much larger than t, in the model where M has random access to them).

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 now need to specify a "reference" complete language relative to which to construct PCPs. As discussed, we choose the language Succinct Circuit Sat (sCSAT for short); informally, sCSAT 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. *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* sCSAT *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 $|F| \le 2^n$.)*

Informally, a PCP system for sCSAT 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 that of 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 polylogarithmic-time verifiers) [BFLS91, BSGH+05, Mie09], we have the following definition:

Definition 2.2. *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 sCSAT** *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 $F \in$ sCSAT,*

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

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 $F \notin$ sCSAT 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 guiding our study of concrete efficiency. Informally, given a PCP system (P, V) for sCSAT, its *concrete-efficiency threshold* B (relative to a cost function $\mathbf{C}$) is the smallest n such that, for every circuit F describing a circuit C of size at least 2^n, the (combined) cost of using the PCP system (P, V) on input F is less than the naive verification overhead of C, when the combined cost is computed with $\mathbf{C}$ taking as input the PCP proving overhead and PCP verification overhead.

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

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

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

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

- the proving time, t_P, is given by $|P_{n,k}|$, which is the size of the relevant prover circuit; while the verification time, t_V, is given by $|Q_{n,k}| + |D_{n,k}|$, because $|Q_{n,k}|$ and $|D_{n,k}|$ are the sizes of the relevant query and decision circuits;
- the ideal proving time, i_P, 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, of size k, on appropriate inputs; while the ideal verification time, i_V, is given by k, because the statement at hand is F and it has size k.

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 $B(\mathbf{C}) = \infty$, when $\mathbf{C}$ is a polynomial cost function. This 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.

2.3 Theorem 2: A B-efficient PCP for sCSAT with $B < \infty$

In Section 1.4 we informally stated our second main theorem. We now give a formal statement of the theorem; its proof is in Section 6.

Theorem 2 (restated)**.** *There exists a PCP system for* sCSAT *(see Definition 2.2) with finite concrete-efficiency threshold (see Definition 2.3) for every polynomial cost function.*

In light of Definition 2.3, it is indeed not clear that such a PCP system exists. For example, if $\frac{|P_{n,|F|}|}{|F| \cdot 2^n} = \Omega(2^n)$, then the cost is $\Omega(2^n)$, and thus $B(\mathbf{C}) = \infty$ for any polynomial cost function $\mathbf{C}$. Similarly if $\frac{|P_{n,|F|}|}{|F| \cdot 2^n} = \Omega(|F|)$ or $\frac{|Q_{n,|F|}| + |D_{n,|F|}|}{|F|} = \Omega(|F|)$.

Thus the efficiency requirements imposed on a PCP system so to fulfill Definition 2.3 with *any finite* concrete-efficiency threshold $B(\mathbf{C})$ (not to mention a small $B(\mathbf{C})$) are already quite stringent.

2.4 Theorem 3: Concrete Efficiency of Proximity Testing to Reed–Solomon

In Section 1.5 we informally discussed the concrete-efficiency threshold of a PCPP system for a code ensemble (see Definition 2) and then informally stated our third main theorem. We now give a formal definition and a formal statement of the theorem.

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

$$\mathbf{C}\left(\frac{n+L(C)}{k}\,,\,Q(C)\right) < k\ . \tag{2}$$

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

Remark 2.8. 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 believe that transitioning to this cleaner information-theoretic definition is justified in our case because, through a suitable use of additive-FFT techniques, we can ensure that the computational costs only account for small additional logarithmic factors.

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, for the more convenient case of domains that are *shifts* of such additive subgroups. 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. *Given a finite field $\mathbb{F}_q$, subset S of $\mathbb{F}_q$, and 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 degree-d polynomials.*

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 $\mathbb{F}_2$-affine-subspace, and $d = |S|/8 - 1$.*

We can now formally restate our third main theorem:

Theorem 3 (restated)**.** *There exists a PCPP system (see Definition 2.5) for the code ensemble $\mathcal{E}_{\mathrm{RS}}$ (see Definition 2.9) having a concrete-efficiency threshold $\hat{B}_{\mathrm{RS}}(\mathbf{C}_\times) \leq 2^{43}$ (see Definition 2.7).*

The proof of Theorem 3, along with a longer discussion of the proof strategy, is given in Section 9.

2.5 Theorem 4: Properties for Cryptographic Constructions

In Section 1.6 we informally stated our fourth main theorem. We now further discuss it. Our Theorem 4 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 three properties:

(A) "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 valid witness by running a knowledge extractor that runs in time proportional to the size of the witness.

Note that we cannot expect the existence of a knowledge extractor that is able to locally decode bits of the witness by having oracle access to a PCP proof. The lack of local decoding here is inherent because our PCPs, as those of [BSS08, BSGH+05], rely on "univariate techniques" (see Remark 12.39). Thus, because we do not achieve a knowledge extractor that is an implicit representation of the valid witness, we call the knowledge property we achieve *explicit* PCP proof of knowledge.

(B) 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 working out these algorithms for the more complex components of the construction is quite non-trivial. We explicitly carry out this separation and provide an algorithmic specification for the query and decision algorithms of our PCP verifier (in addition to the algorithmic specification of the "non-split" verifier already contained Appendix B). This provides a reference for others wishing to study the concrete efficiency of cryptographic constructions where the query and decision algorithms are invoked.

(C) 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.[11]

Using our PCPs in cryptographic constructions. The explicit PCP proof of knowledge mentioned above falls short of the one needed in the construction of universal arguments [BG08], where a knowledge extractor that is able to perform local decoding of the witness is needed. Nonetheless, we show that the same construction as [BG08, Construction 3.4] still gives us succinct interactive arguments of knowledge for NP.

More precisely, our result is stronger than just stated: we obtain universal arguments where the knowledge extractor may run in time that is *polynomial* (rather than polylogarithmic as in [BG08]) in the witness length — we call these *almost universal arguments*. (In particular, for example, with a mild superpolynomial-hardness assumption, we can obtain universal arguments for NP.) Once again, this limitation is only natural in light of the fact that our PCPs can only satisfy an "explicit" proof-of-knowledge property, with no local decoding of the witness.

Nonetheless, the aforementioned limitation is actually not a problem because succinct arguments of knowledge for NP suffice for most "positive" applications of PCPs, where the relevant computations to be verified anyways "lie in NP" (see Remark 12.2).

The properties that we use in the construction of almost universal arguments, following [BG08, Construction 3.4], are a superset of the properties required of PCPs in other cryptographic applications such as [Mic00, Val08, CT10, BCCT12a, CT12]. Thus, in the proof, it will suffice to focus on the construction of almost universal arguments.

The proof of Theorem 4 is given in Section 12.

3 Open Questions

We briefly discuss intriguing questions about the computational efficiency of PCPs that our work leaves unanswered.

Complexity-preserving PCPs. Our Theorem 1 guarantees the existence of a PCP system in which the prover and verifier respectively run in time $(|\mathbb{P}| + T) \cdot \mathsf{polylog}(T)$ and $(|\mathbb{P}| + |\mathsf{x}|) \cdot$

[11] This fact was already by Mie [Mie09] for a special case of the proximity testers underlying our PCPs; he used their efficient reverse samplability for showing a verifier-efficient construction of gap amplification [Din07].

$\mathsf{polylog}(T)$, when proving and verifying that a program $\mathbb{P}$ accepts $(\mathtt{x}, \mathtt{w})$ for some $\mathtt{w}$. These running times are quasi-optimal. However, the running time of the prover is achieved via the use of FFT-like methods, which means that its space complexity is $\Omega(T)$ *regardless* of the space complexity of $\mathbb{P}$. Let us define a PCP system to be *complexity preserving* if it has the aforementioned running times and, in addition, the prover runs in space $(|M|+|\mathtt{x}|+S)\cdot\mathsf{polylog}(T)$ when $\mathbb{P}$ uses at most space S. Do complexity-preserving PCPs exist? Bitansky and Chiesa [BC12] have constructed a complexity-preserving multi-prover interactive proof, but it is unclear whether complexity-preserving PCPs can be constructed or not.

Limits of concrete efficiency. We have introduced new efficiency measures for PCPs that attempt to capture how "soon" a PCP becomes useful, and have shown progress relative to these complexity measures. How far can the concrete efficiency of PCPs be pushed?

A natural starting point would be to develop an even more precise understanding of the Bivariate Testing Theorem of Polishchuk and Spielman [PS94], e.g., by understanding what is the *optimal* value of the universal constant of bivariate testing. Concretely, we believe that Proposition 9.2 can be further improved and it is an important open problem to do so: even small improvements (e.g., relaxing the requirement in the statement 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.

We are optimistic that additional work in this direction will discover surprisingly efficient PCP constructions.

Local decoding of PCPs. Our Theorem 4 states that the PCPs from our Theorem 1 can be used to construct a weaker variant of universal arguments [BG08]. As discussed, the single reason why we do not obtain (full) universal arguments is that our PCPs, being based on techniques that use univariate polynomials, do not enjoy good local decoding properties. Are there universal arguments where the prover and verifier running times are quasi-optimal? Bitansky and Chiesa [BC12] use their complexity-preserving MIP construction to obtain complexity-preserving universal arguments *with private coins*, but the public-coin case (often crucial in applications [Bar01, PR05]) remains open (even without posing requirements on the space complexity of the prover).

4 Definitions

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

4.1 Reed–Solomon and Reed–Muller Codes

Definition 4.1. *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 4.2. *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 4.3. *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 4.4. *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 4.5. *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 4.6. *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*

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

4.2 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 4.7. *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 4.8. *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 4.9. *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 4.10. *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 4.11. *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 4.12 (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.*

4.3 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 4.13. *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 4.14. *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 4.15. *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.[12] (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.)

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

5 Proof of Theorem 1

In this section we prove Theorem 1, discussed in Section 1.2 and formally stated in Section 2.1.

High-level strategy. It is useful to break the problem of constructing PCPs for $\mathrm{BH}_{\mathsf{RAM}}$ with stringent efficiency requirements into *two* different subproblems: (a) constructing a very efficient PCP system for some "PCP-friendly" NEXP-complete problem $\mathcal{P}$, and (b) constructing tight reductions from $\mathrm{BH}_{\mathsf{RAM}}$ to $\mathcal{P}$. Indeed, PCPs are only known to exist for certain problems having strong algebraic or combinatorial structure. 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* (sACSP) 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 $\mathrm{BH}_{\mathsf{RAM}}$).[13] In other words, we choose $\mathcal{P} :=$ sACSP. This choice "decouples" (a) and (b), and progress can be made independently on each subproblem.[14] In light of the aforementioned decoupling, our proof of Theorem 1 relies on two ingredients: (a) an algebraic PCP construction for sACSP, and (b) the use of reductions of Ben-Sasson et al. [BSCGT13] from $\mathrm{BH}_{\mathsf{RAM}}$ to sACSP.

The family of succinct algebraic constraint satisfaction problems sACSP is introduced and defined in Section 7. Following is the proof of Theorem 1.

Proof of Theorem 1. For a random-access machine M, let $|M|$ denote the size of the transition function of M when viewed as a Boolean circuit and $\deg(M)$ the total degree of this circuit when viewed as a polynomial. We begin by establishing the following claim:

Claim 5.1. *There is a PCP system $(P'_{\mathrm{RAM}}, V'_{\mathrm{RAM}})$ where, in order to verify that a random-access machine M accepts $(\mathtt{x}, \mathtt{w})$, for some $\mathtt{w}$, within T steps (with $|\mathtt{x}|, |\mathtt{w}| \leq T$),*

- *the prover runs in sequential time $|M| \cdot T \cdot \mathsf{polylog}(T) \cdot \deg(M)$ and parallel time $O((\log T + \log\deg(M))^2)$ when given a transcript of computation of M on $(\mathtt{x}, \mathtt{w})$, and*
- *the verifier in sequential time $(|M| + |\mathtt{x}|) \cdot \mathsf{polylog}(T + \deg(M))$ and parallel time $O((\log T + \log\deg(M))^2)$.*

Proof. Ben-Sasson et al. [BSCGT13] study reductions from *bounded-halting problems on random-access machines*, denoted $\mathrm{BH}_{\mathsf{RAM}}$, to sACSP. Specifically, for a given random-access machine M, $\mathrm{BH}_{\mathsf{RAM}}(M)$ is the language of pairs $(\mathtt{x}, T)$ such that the random-access machine M nondeterministically accepts $\mathtt{x}$ within T steps (with $|\mathtt{x}| \leq T$). It is convenient to assume without loss of generality that all instances $(\mathtt{x}, T)$ have T that is a positive power of 2, so that we can represent an instance $(\mathtt{x}, 2^t)$ via the string $(\mathtt{x}, 1^t)$.

A reduction from $\mathrm{BH}_{\mathsf{RAM}}$ to sACSP is a pair (Φ, Ψ) of efficient transformations such that:

[13] We shall thus focus on *algebraic PCPs*. The reason is that 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]. (Also see Remark 8.10.)

[14] 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.

- For every random-access machine M,

$$\mathsf{par}^M := \Phi(M) = \Big(f, (\mathsf{m_H}, \mathsf{t_H}, \mathbf{H}), (\mathsf{c_N}, \mathsf{t_N}, \mathsf{s_N}, \mathbf{N}), (\mathsf{t_P}, \mathsf{s_P}, \mathbf{P}), (\mathsf{t_I}, \mathbf{I})\Big)$$

  is a choice of parameters for sACSP; in other words, par^M specifies, for each positive integer t, what algebraic constraints are to be used to verify membership of instances of the form $(\mathtt{x}, 1^t)$. (See Definition 7.1 in Section 7 for the formal definition of sACSP.)

- For every random-access machine M and instance $(\mathtt{x}, 1^t)$, it holds that $(\mathtt{x}, 1^t) \in \mathrm{BH}_{\mathsf{RAM}}(M)$ if and only if $(\mathtt{x}, 1^t) \in \mathrm{sACSP}(\mathsf{par}^M)$.

- For every random-access machine M and instance $(\mathtt{x}, 1^t)$, if $\mathtt{w}$ is a witness to the fact that $(\mathtt{x}, 1^t) \in \mathrm{BH}_{\mathsf{RAM}}(M)$, then $A := \Psi(M, \mathtt{w})$ is a witness to the fact that $(\mathtt{x}, 1^t) \in \mathrm{sACSP}(\mathsf{par}^M)$.

At high level, the construction of the PCP system $(P'_{\mathrm{RAM}}, V'_{\mathrm{RAM}})$ consists of first invoking the reductions of Ben-Sasson et al. [BSCGT13] from $\mathrm{BH}_{\mathsf{RAM}}$ to sACSP and then invoking the PCP system $(P_{\mathrm{sACSP}}, V_{\mathrm{sACSP}})$ for sACSP guaranteed by our Theorem 8.1 from Section 8.[15] More precisely, the PCP prover P'_{RAM} and the PCP verifier V'_{RAM} work as follows.

- P'_{RAM}, on input a random-access machine M, instance $(\mathtt{x}, 1^t)$, and witness $\mathtt{w}$ such that $M(\mathtt{x}, \mathtt{w})$ accepts within 2^t, performs the following steps:
  1. compute $\mathsf{par}^M := \Phi(M)$;
  2. compute $A := \Psi(M, \mathtt{w})$;
  3. compute $\pi := P_{\mathrm{sACSP}}\big((\mathtt{x}, 1^t), A\big)$, with choice of parameters par^M;
  4. output π.

- V'_{RAM}, on input a random-access machine M and instance $(\mathtt{x}, 1^t)$, and with oracle access to a PCP oracle π, performs the following steps:
  1. compute $\mathsf{par}^M := \Phi(M)$;
  2. compute $b := V^{\pi}_{\mathrm{sACSP}}\big((\mathtt{x}, 1^t)\big)$, with choice of parameters par^M;
  3. output b.

Due to the soundness guarantees of the reduction Φ, $(P'_{\mathrm{RAM}}, V'_{\mathrm{RAM}})$ is a PCP system for bounded-halting problems on random-access machines. We are thus only left to argue its efficiency.

The reductions of Ben-Sasson et al. [BSCGT13] generate choices of parameters par^M that are "natural", so that Theorem 8.1 ensures that P_{sACSP} runs in time $\mathsf{s_P}(1^t) \cdot 2^{f(t)} \cdot \mathsf{poly}(f(t))$ and V_{sACSP} runs in time $\big(|\mathtt{x}| + \mathsf{s_P}(1^t)\big) \cdot \mathsf{poly}(f(t))$. (See Remark 8.2.) Furthermore, in Ben-Sasson et al. [BSCGT13], it holds that $\mathsf{s_P}(1^t) = |M| \cdot \mathsf{poly}(f(t))$ and $2^{f(t)} = 2^t \cdot \mathsf{polylog}(2^t) \cdot \deg(M)$. Putting these two facts together yields the desired sequential efficiency for $(P'_{\mathrm{RAM}}, V'_{\mathrm{RAM}})$. The parallel efficiency can be similarly argued. □

We now return to the proof of Theorem 1. We can fix a universal random-access machine U where both $|U|$ and $\deg(U)$ are on the order of $\mathsf{polylog}(T)$, and then simulate a given random-access machine M (computing on a given $(\mathtt{x}, \mathtt{w})$) on U step by step. The overhead at each step

[15] Ultimately, when proving Theorem 4, we will in fact need *Levin* reductions, where we are additionally guaranteed the existence of an "inverse" to the witness reduction Ψ, so to ensure that the knowledge property of our PCP system $(P_{\mathrm{sACSP}}, V_{\mathrm{sACSP}})$ is preserved by the reductions. The reductions of Ben-Sasson et al. [BSCGT13] are Levin reductions, so this is not a problem; see the proof of Theorem 4 and Remark 12.1 in Section 12.

is polynomial in the register sizes, which is only $O(\log T)$. Thus, overall, simulating M on U is only $\mathsf{polylog}(T)$ slower than running M directly. Combining this with Claim 5.1 yields a PCP system $(P_{\text{RAM}}, V_{\text{RAM}})$ satisfying the properties claimed in the theorem statement. $\square$

6 Proof of Theorem 2

In this section we prove Theorem 2, discussed in Section 1.4 and formally stated in Section 2.3.

Proof sketch of Theorem 2. To construct a PCP system for sCSAT with $B(\mathbf{C}) < \infty$ for any polynomial cost function $\mathbf{C}$, it suffices to construct a PCP system for sCSAT where the prover runs in time $|F| \cdot 2^n \cdot \mathsf{poly}(n)$ and the verifier runs in time $|F| \cdot \mathsf{poly}(n)$. So let M^* be the random-access machine that, on input a circuit descriptor F, will non-deterministically verify that $F \in$ sCSAT; note that this can be done in time $T := O(|F| \cdot 2^n)$ when F describes a circuit of size 2^n. We can now use the PCP system from Theorem 1 on the random-access machine M^*. Doing so yields a PCP system for sCSAT where the prover runs in time $|F| \cdot 2^n \cdot \mathsf{poly}(n)$ and the verifier runs in time $|F| \cdot \mathsf{poly}(n)$, as desired. $\square$

Remark 6.1 (are RAMs necessary?)**.** In the proof of Theorem 2 we relied on Theorem 1, whose proof, among other things, relies on fast reductions from random-access machines. We note that attempting to prove Theorem 2 via a "direct" reduction from sCSAT to sACSP through a naive use of the techniques contained in [BSCGT13] will not work. Indeed, if we were to simply route the circuit represented by F using De Bruijn graphs (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 the random-access machine M^* that 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 path is unnecessarily complicated, and taking a path, as we do, through random-access machines is more natural and cleaner.

7 A Succinct Algebraic Constraint Satisfaction Problem

sACSP is a class of succinct algebraic constraint satisfaction problems, each specified by a list of parameters, for univariate polynomials over finite field extensions of GF(2). Roughly, sACSP is simply a class of succinct graph-coloring problems that must respect certain algebraic constraints.

Each particular sACSP problem simultaneously allows for the construction of PCPs with quasilinear-time prover and polylogarithmic-time 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 by Ben-Sasson et al. [BSCGT13].

Informally, a choice par of parameters of sACSP 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 an affine subspace of $\mathbb{F}_T$.
- A family $\{\vec{N}_T\}_{T\in\mathbb{N}}$, where each $\vec{N}_T$ is a vector of neighbor polynomials.
- A family $\{P_T\}_{T\in\mathbb{N}}$, where each P_T is a constraint polynomial.
- A family $\{\vec{I}_T\}_{T\in\mathbb{N}}$, where each $\vec{I}_T$ is a vector of affine subspaces each contained in H_T.

A certain algebraic relation constrains which parameters are possible. A reduction to sACSP will concretely instantiate a choice of the above parameters.

A pair (x, T) is a member of the language $\mathrm{sACSP}(\mathsf{par})$ 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) for every element α of the subspace H_T, the constraint polynomial P_T, when given as input the colors in the "$\vec{N}_T$-induced neighborhood" of α, is satisfied; and (2) the colors of elements in the $(\log|\mathsf{x}|)$-th affine subspace in $\vec{I}_T$ are consistent with x.

Crucially, for each of the families in par, the T-th object must be able to be "understood" in time $\mathsf{polylog}(T)$ (e.g., generating an element in H_T, generating an arithmetic circuit for each polynomial in $\vec{N}_T$, etc.); this is the requirement of *succinctness* of the problem. Additional properties that are essential for us to construct PCPs with a quasilinear-time prover and polylogarithmic-time verifier include, for example, the fact that H_T is an affine subspace (so that one may leverage the computational properties of *linearized polynomials* [LN97, Section 2.5]).

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 $T \approx 2^t$.

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 $\left\{\mathbb{F}_t\right\}_{t\in\mathbb{N}}$ 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 $\textsc{FindIrrPoly}(1^{f(t)})$;
2. two proper functions associated with the family $\mathbf{H}$ in Parameter 3:
   (a) a dimension function $\mathsf{m}_{\mathbf{H}}\colon \mathbb{N} \to \mathbb{N}$, and
   (b) a time function $\mathsf{t}_{\mathbf{H}}\colon \mathbb{N} \to \mathbb{N}$.

3. a family $\mathbf{H} = \{H_t\}_{t \in \mathbb{N}}$ such that:
   (a) H_t is an $\mathsf{m}_{\mathbf{H}}(t)$-dimensional affine subspace of $\mathbb{F}_t$ specified by a basis $\mathcal{B}_{H_t}$ and an offset $\mathcal{O}_{H_t}$ for all $t \in \mathbb{N}$, and
   (b) there exists a $\mathsf{t}_{\mathbf{H}}$-time algorithm $\textsc{Find}\mathbf{H}$ such that $(\mathcal{B}_{H_t}, \mathcal{O}_{H_t}) = \textsc{Find}\mathbf{H}(1^t)$ for all $t \in \mathbb{N}$;
4. three proper functions associated with the family $\mathbf{N}$ in Parameter 5:
   (a) a 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 size function $\mathsf{s}_{\mathbf{N}} \colon \mathbb{N} \to \mathbb{N}$;
5. a family $\mathbf{N} = \{\vec{N}_t\}_{t \in \mathbb{N}}$ such that:
   (a) $\vec{N}_t = (N_{t,i} \colon \mathbb{F}_t \to \mathbb{F}_t)_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)}$ is a vector of $\mathsf{c}_{\mathbf{N}}(t)$ neighbor polynomials over $\mathbb{F}_t$, and
   (b) there exists a $\mathsf{t}_{\mathbf{N}}$-time algorithm $\textsc{Find}\mathbf{N}$ such that $[N_{t,i}]^{\mathsf{A}} = \textsc{Find}\mathbf{N}(1^t, i)$ is an $\mathsf{s}_{\mathbf{N}}$-size $\mathbb{F}_t$-arithmetic circuit computing $N_{t,i}$ for all $t \in \mathbb{N}$ and $i \in \{1, \dots, \mathsf{c}_{\mathbf{N}}(t)\}$;
6. two proper functions associated with the family $\mathbf{P}$ in Parameter 7:
   (a) a time function $\mathsf{t}_{\mathbf{P}} \colon \mathbb{N} \to \mathbb{N}$, and
   (b) a size function $\mathsf{s}_{\mathbf{P}} \colon \mathbb{N} \to \mathbb{N}$;
7. 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, and
   (b) there exists a $\mathsf{t}_{\mathbf{P}}$-time algorithm $\textsc{Find}\mathbf{P}$ such that $[P_t]^{\mathsf{A}} = \textsc{Find}\mathbf{P}(1^t)$ is a $\mathsf{s}_{\mathbf{P}}$-size $\mathbb{F}_t$-arithmetic circuit computing P_t for all $t \in \mathbb{N}$;
8. a proper function associated with the family $\mathbf{I}$ in Parameter 9:
   (a) a time function $\mathsf{t}_{\mathbf{I}} \colon \mathbb{N} \to \mathbb{N}$.
9. a family $\mathbf{I} = \{\vec{I}_t\}_{t \in \mathbb{N}}$ such that:
   (a) $\vec{I}_t = (I_{t,m})_{m=1}^{t}$ is a vector of affine subspaces each contained in H_t specified by a basis $\mathcal{B}_{I_{t,m}}$ of m elements and an offset $\mathcal{O}_{I_{t,m}}$ for all $t \in \mathbb{N}$, and
   (b) there exists a $\mathsf{t}_{\mathbf{I}}$-time algorithm $\textsc{Find}\mathbf{I}$ such that $(\mathcal{B}_{I_{t,m}}, \mathcal{O}_{I_{t,m}}) = \textsc{Find}\mathbf{I}(1^t, 1^m)$ for all $t \in \mathbb{N}$ and $m \in \{1, \dots, t\}$.

The following **algebraic constraint** must hold:

$$\forall\, t \in \mathbb{N}\,,\ \deg\left(P_t\left(x, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \dots, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathsf{c}_{\mathbf{N}}(t)})}\right)\right) \le 2^{f(t)-2}\ . \tag{3}$$

The language sACSP, with respect to a choice $\mathsf{par}_{\mathrm{sACSP}}$

$$\mathsf{par}_{\mathrm{sACSP}} = \Big(f, (\mathsf{m}_{\mathbf{H}}, \mathsf{t}_{\mathbf{H}}, \mathbf{H}), (\mathsf{c}_{\mathbf{N}}, \mathsf{t}_{\mathbf{N}}, \mathsf{s}_{\mathbf{N}}, \mathbf{N}), (\mathsf{t}_{\mathbf{P}}, \mathsf{s}_{\mathbf{P}}, \mathbf{P}), (\mathsf{t}_{\mathbf{I}}, \mathbf{I})\Big)$$

of the above parameters, consists of instances $(\mathtt{x}, 1^t)$, where $\mathtt{x}$ is a binary input string and $t \in \mathbb{N}$ with $|\mathtt{x}| \in \{2, \dots, 2^t\}$, 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 $\alpha \in H_t$,

$$P_t\Big(\alpha, \big(A \circ N_{t,1}\big)(\alpha), \ldots, \big(A \circ N_{t,\mathbf{c_N}(t)}\big)(\alpha)\Big) = 0_{\mathbb{F}_t} \ .$$

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

(ii) *Consistency with the input.* Letting α_i be the i-th element in $I_{t,\log|\mathtt{x}|}$ for every index $i \in \{1, \ldots, |\mathtt{x}|\}$,

$$\mathtt{x} = \mathrm{bit}\big(A(\alpha_1)\big) \cdots \mathrm{bit}\big(A(\alpha_{|\mathtt{x}|})\big) \ ,$$

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 input $\mathtt{x}$.

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] 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 an affine subspace) because their paper does not attempt to construct PCP verifiers that are succinct in time. (Also 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.[16]

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.

[16] A technical difference of our Definition 7.1 compared to [BSGH$^+$05, Definition 6.1] that is *not* a generalization is the form of the instance consistency check: in Definition 7.1, the instance consistency check is taken to be of a very specific form, namely, equality. This restriction simplifies our PCP construction for sACSP (discussed in Section 8). While we could construct (somewhat more complex) PCPs to account for an instance consistency check of a more general form (see Remark 8.7), we do not believe that such a gain in generality is justified, because this degree of freedom is not exploited in the reductions of Ben-Sasson et al. [BSCGT13].

8 A PCP for sACSP

We denote by V_{RS} and V_{VRS} two PCPP verifiers that respectively test proximity to RS and VRS over affine subspaces of finite field extensions of $\mathbb{F}_2$; we denote by P_{RS} and P_{VRS} their corresponding PCPP provers.

In this section we show how to use these four algorithms to construct a PCP system $(P_{\mathrm{sACSP}}, V_{\mathrm{sACSP}})$ for the language $\mathrm{sACSP}(\mathsf{par})$, for a given choice par of parameters.

When this construction is instantiated with our constructions of V_{RS} and V_{VRS} (see Algorithm 13 and Algorithm 12 and their soundness analysis in Section 11) and P_{RS} and P_{VRS} (see Algorithm 4 and Algorithm 3), we obtain a PCP system with quasilinear-time prover and succinct verifier (that only depends quasilinearly on the input instance). This PCP system is used in the proof of Theorem 1 in Section 5.

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 sACSP, for the case where the verifier does 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 $\mathrm{sACSP}(\mathsf{par})$, which involves probabilistically-checking properties of univariate polynomials with huge degrees.[17] Moreover, we also need to ensure that the verifier does not perform expensive computations on F, the "explicit" input (circuit descriptor) 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 2.4) 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 $\mathrm{sACSP}(\mathsf{par})$ where, roughly, the prover runs in $T \cdot \mathsf{poly}(T)$ time and the verifier runs in $|\mathsf{x}| \cdot \mathsf{polylog}(T)$ time. The definition of sACSP will ensure that the verifier is able to succinctly generate appropriate representations for all the relevant objects (such as a small arithmetic circuit computing 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.

[17] 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]).

The construction is roughly as follows.

First recall from Section 7 that in order to verify whether a given instance $(\mathtt{x}, T)$ is a member in $\mathrm{sACSP}(\mathsf{par})$, for a given choice of parameters par, one needs to establish whether there is a low-degree assignment polynomial A that satisfies two conditions: (i) when "composed" with the constraint polynomial P_T, A vanishes everywhere on H_T; and, moreover, (ii) A is "consistent" with $\mathtt{x}$.

To check the first condition, the PCP verifier V_{sACSP} works as follows. The verifier V_{sACSP} asks the prover to provide an evaluation p_0 of A along with a proximity proof π_0, in order to test proximity of p_0 to the appropriate Reed–Solomon code by invoking V_{RS}. The verifier V_{sACSP} also asks the prover to provide an evaluation p_1 of B, the polynomial that one obtains when composing A with P_T, along with a proximity proof p_1, in order to test proximity of p_1 to the appropriate *Vanishing* Reed–Solomon code on the affine subspace H_T by invoking V_{VRS}. After this, the verifier V_{sACSP} 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, the verifier V_{sACSP} works as follows. The verifier V_{sACSP} asks the prover to further provide a proximity proof π_{c} for the proximity of $p_0 - p_{\mathtt{x}}$ to an appropriate Vanishing Reed–Solomon code on the $(\log|\mathtt{x}|)$-th affine subspace in $\vec{I}_T$, where $p_{\mathtt{x}}$ is a function depending on $\mathtt{x}$ that can be easily evaluated by the verifier. Intuitively, this will ensure that p_0 takes on the bits of $\mathtt{x}$, as required, on the $(\log|\mathtt{x}|)$-th affine subspace in $\vec{I}_T$. (In our case, $\mathtt{x}$ 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 an affine subspace is important to ensure that the prover and verifier can run in time quasilinear in the instance size.

Thus, the PCP prover P_{sACSP}, 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 the polynomial B computed from A and P_T is of low-enough degree and vanishes on H_T; 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 $p_0 - p_{\mathtt{x}}$, where $p_{\mathtt{x}}$ is a function depending on the input $\mathtt{x}$, is of low-enough degree and vanishes on the $(\log|\mathtt{x}|)$-th affine subspace in $\vec{I}_T$.

8.2 The Construction In Detail

Formally, we prove the following theorem:

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

$$\mathsf{par} = \Big(f, (\mathsf{m}_{\mathbf{H}}, \mathsf{t}_{\mathbf{H}}, \mathbf{H}), (\mathsf{c}_{\mathbf{N}}, \mathsf{t}_{\mathbf{N}}, \mathsf{s}_{\mathbf{N}}, \mathbf{N}), (\mathsf{t}_{\mathbf{P}}, \mathsf{s}_{\mathbf{P}}, \mathbf{P}), (\mathsf{t}_{\mathbf{I}}, \mathbf{I})\Big) \enspace .$$

There exists a PCP system $(P_{\text{sACSP}}, V_{\text{sACSP}})$ for $\mathrm{sACSP}(\mathsf{par})$ *with perfect completeness and soundness* $1/2$*. Moreover:*

- P_{sACSP} *runs in sequential time*

$$\mathsf{setup}_s(1^t) + \left(|\mathrm{x}| + \left(\mathsf{s}_{\mathbf{P}}(1^t) + \sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \mathsf{s}_{\mathbf{N}}(1^t, i) \right) \cdot 2^{f(t)} \right) \cdot \mathsf{poly}(f(t))$$

  and parallel time $\mathsf{setup}_p(1^t) + O(f(t)^2)$.

- V_{sACSP} *runs in sequential time*

$$\mathsf{setup}_s(1^t) + \left(|\mathrm{x}| \cdot \left(\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}) \right) + \left(\mathsf{s}_{\mathbf{P}}(1^t) + \sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \mathsf{s}_{\mathbf{N}}(1^t, i) \right) \right) \cdot \mathsf{poly}(f(t))$$

  and parallel time $\mathsf{setup}_p(1^t) + O(f(t)^2)$.

Above, $\mathsf{setup}_s(1^t)$ *is the setup sequential time to instantiate the parameters with index* t *and equals*

$$\mathsf{time}_{\text{FindIrrPoly}}(f(t)) + \mathsf{t}_{\mathbf{H}}(1^t) + \left(\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \mathsf{t}_{\mathbf{N}}(1^t, i) \right) + \mathsf{t}_{\mathbf{P}}(1^t) + \mathsf{t}_{\mathbf{I}}(1^t, 1^{\log |\mathrm{x}|}) \enspace ,$$

where $\mathsf{time}_{\text{FindIrrPoly}}(f(t))$ *is the time for generating an irreducible polynomial of degree* $f(t)$ *(see Algorithm 17);* $\mathsf{setup}_p(1^t)$ *is the corresponding setup parallel time.*

Remark 8.2. For "natural" choices of parameters par (e.g., those generated by the reductions of Ben-Sasson et al. [BSCGT13] from random-access machines),

$$\begin{aligned} \mathsf{setup}_s(1^t) &= \mathsf{s}_{\mathbf{P}}(1^t) \cdot \mathsf{poly}(f(t)) \\ \mathsf{setup}_p(1^t) &= O(f(t)^2) \enspace , \\ \sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}) &= \mathsf{poly}(f(t)) \enspace , \\ \sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \mathsf{s}_{\mathbf{N}}(1^t, i) &= \mathsf{poly}(f(t)) \enspace , \end{aligned}$$

In such a case, recalling that $|\mathrm{x}| \leq 2^t \leq 2^{f(t)}$,

- P_{sACSP} runs in sequential time $\mathsf{s}_{\mathbf{P}}(1^t) \cdot 2^{f(t)} \cdot \mathsf{poly}(f(t))$ and parallel time $O(f(t)^2)$,
- V_{sACSP} runs in sequential time $\big(|\mathrm{x}| + \mathsf{s}_{\mathbf{P}}(1^t)\big) \cdot \mathsf{poly}(f(t))$ and parallel time $O(f(t)^2)$.

Moreover, the degree of the field extension $f(t)$ (which induces the corresponding field size $2^{f(t)}$) is the main measure of complexity, since usually $\mathsf{s}_{\mathbf{P}}(1^t) = \mathsf{poly}(f(t))$.

We recall that in Appendix B we give a *complete and detailed* algorithmic specification for $(P_{\text{sACSP}}, V_{\text{sACSP}})$. To the best of our knowledge, this is the first explicit construction of a PCP system with short proofs (and, in our case, fast provers) and polylogarithmic-time verifiers, and provides a valuable starting point for studying the efficiency of PCPs from a practical perspective.[18]

In this section, we give the construction of $(P_{\text{sACSP}}, V_{\text{sACSP}})$, in terms of V_{RS} and V_{VRS} and their corresponding provers P_{RS} and P_{VRS}, and then prove its completeness and soundness properties. (More precisely, we shall use the "amplified versions" V_{aRS} and V_{aVRS} of V_{RS} and V_{VRS} that allow us to control the proximity and soundness parameters; see Remark 11.2, and Algorithm 11 and Algorithm 10 for more details.)

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

- in Construction 8.3 we describe the PCP prover P_{sACSP},
- in Definition 8.4 we define the "format" of the PCP oracle, and
- in Construction 8.5 we describe the PCP verifier V_{sACSP}.

Concise pseudo-code listings for the PCP prover P_{sACSP} and PCP verifier V_{sACSP} can also be found in Algorithm 2 and Algorithm 9 respectively.

The PCP prover P_{sACSP} for sACSP is as follows:

Construction 8.3 (PCP Prover for sACSP)**.** The PCP prover for sACSP, with respect to a choice

$$\mathsf{par} = \Big(f, (\mathsf{m}_{\mathbf{H}}, \mathsf{t}_{\mathbf{H}}, \mathbf{H}), (\mathsf{c}_{\mathbf{N}}, \mathsf{t}_{\mathbf{N}}, \mathsf{s}_{\mathbf{N}}, \mathbf{N}), (\mathsf{t}_{\mathbf{P}}, \mathsf{s}_{\mathbf{P}}, \mathbf{P}), (\mathsf{t}_{\mathbf{I}}, \mathbf{I})\Big)$$

of parameters, is denoted P_{sACSP} and has hard-coded in it this choice of parameters. On input an instance $(\mathtt{x}, 1^t)$ and an assignment polynomial $A \colon \mathbb{F}_t \to \mathbb{F}_t$ that is a witness to the membership of $(\mathtt{x}, 1^t)$ in sACSP (see Definition 7.1), the PCP prover P_{sACSP} 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.) 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 and offset for the affine subspace H_t of $\mathbb{F}_t$ by computing $(\mathcal{B}_{H_t}, \mathcal{O}_{H_t}) := \textsc{Find}\mathbf{H}(1^t)$. (See Parameter 3 in Definition 7.1.)
   (c) Find the polynomials $(N_{t,i}(x))_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)}$ that induce the "neighborhood" of each element in $\mathbb{F}_t$: for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, compute $[N_{t,i}]^{\mathsf{A}} := \textsc{Find}\mathbf{N}(1^t, i)$. (See Parameter 5 in Definition 7.1.)
   (d) Find the constraint polynomial by computing $[P_t]^{\mathsf{A}} := \textsc{Find}\mathbf{P}(1^t)$. (See Parameter 7 in Definition 7.1.)
   (e) Find the basis and offset for the $(\log|\mathtt{x}|)$-th affine subspace $I_{t,\log|\mathtt{x}|}$ in $\vec{I_t}$ by computing $(\mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}) := \textsc{Find}\mathbf{I}(1^t, 1^{\log|\mathtt{x}|})$. (See Parameter 9 in Definition 7.1.)
2. *Generate a proof that A has low degree:*

[18] 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 in Appendix B 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 C.

(a) Let p_0 be the evaluation table of A on $\mathbb{F}_t$:

$$p_0 := \Big\{ \big(\alpha, A(\alpha)\big) \,:\, \alpha \in \mathbb{F}_t \Big\} \enspace .$$

(b) Compute a proof of proximity π_0 for p_0 to $\mathrm{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$:

$$\pi_0 := P_{\mathrm{RS}}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, A\big) \enspace .$$

3. *Generate a proof that A satisfies the constraint polynomial:*
   (a) Define the polynomial $B(x) := P_t\big(x, A(N_{t,1}(x)), \dots, A(N_{t,\mathsf{c}_{\mathbf{N}}(t)}(x))\big)$.
   (b) Let p_1 be the evaluation table of B on $\mathbb{F}_t$:

$$p_1 := \Big\{ \big(\alpha, B(\alpha)\big) \,:\, \alpha \in \mathbb{F}_t \Big\} \enspace .$$

   (c) Define $d := \deg(P_t(x, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \dots, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathsf{c}_{\mathbf{N}}(t)})}))$.
   (d) Compute a proof of proximity π_1 for p_1 to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d)$:

$$\pi_1 := P_{\mathrm{VRS}}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{H_t}, \mathcal{O}_{H_t}, d, B\big) \enspace .$$

4. *Generate a proof that A is consistent with $(\mathtt{x}, 1^t)$:*
   (a) Define $A_{\mathtt{x}}$ to be the low-degree extension of the function $f_{\mathtt{x}} \colon I_{t,\log|\mathtt{x}|} \to \{0,1\}$ defined by $f_{\mathtt{x}}(\alpha_i) := \mathtt{x}_i$ where α_i is the i-th element in $I_{t,\log|\mathtt{x}|}$.
   (b) Define $A' := A - A_{\mathtt{x}}$.
   (c) Compute a proof of proximity π_{c} for the evaluation of A' to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, I_{t,\log|\mathtt{x}|}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$:

$$\pi_{\mathsf{c}} := P_{\mathrm{VRS}}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, A'\big) \enspace .$$

5. Set $\pi := (p_0, \pi_0, p_1, \pi_1, \pi_{\mathsf{c}})$ and output π.

The PCP prover P_{sACSP} from Construction 8.3 allows us to define the format of a PCP proof for membership into the language sACSP:

Definition 8.4 (PCP Proof for sACSP)**.** *A PCP proof of membership into* sACSP*, with respect to a choice*

$$\mathsf{par} = \Big(f, (\mathsf{m}_{\mathbf{H}}, \mathsf{t}_{\mathbf{H}}, \mathbf{H}), (\mathsf{c}_{\mathbf{N}}, \mathsf{t}_{\mathbf{N}}, \mathsf{s}_{\mathbf{N}}, \mathbf{N}), (\mathsf{t}_{\mathbf{P}}, \mathsf{s}_{\mathbf{P}}, \mathbf{P}), (\mathsf{t}_{\mathbf{I}}, \mathbf{I})\Big)$$

of parameters, for an instance $(\mathtt{x}, 1^t)$ 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 $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d)$ where the degree bound d is defined by $d := \deg(P_t(x, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \dots, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathsf{c}_{\mathbf{N}}(t)})}))$, and*
5. *π_{c} is a proof of proximity for $p_0 - p_{\mathtt{x}}$ to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, I_{t,\log|\mathtt{x}|}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$, where $p_{\mathtt{x}}$ is the low-degree extension of $\mathtt{x}$ over $I_{t,\log|\mathtt{x}|}$ and $I_{t,\log|\mathtt{x}|}$ is the $(\log|\mathtt{x}|)$-th affine subspace in $\vec{I}_t$.*

The PCP verifier V_{sACSP} for sACSP is as follows:

Construction 8.5 (PCP Verifier for sACSP)**.** The PCP verifier for sACSP, with respect to a choice

$$\mathsf{par} = \Big(f, (\mathsf{m}_{\mathbf{H}}, \mathsf{t}_{\mathbf{H}}, \mathbf{H}), (\mathsf{c}_{\mathbf{N}}, \mathsf{t}_{\mathbf{N}}, \mathsf{s}_{\mathbf{N}}, \mathbf{N}), (\mathsf{t}_{\mathbf{P}}, \mathsf{s}_{\mathbf{P}}, \mathbf{P}), (\mathsf{t}_{\mathbf{I}}, \mathbf{I})\Big)$$

of parameters, is denoted V_{sACSP} and has hard-coded in it this choice of parameters. On input an instance $(\mathtt{x}, 1^t)$ and with oracle access to a proof π following the format of Definition 8.4, the verifier V_{sACSP} 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.) 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 and offset for the affine subspace H_t of $\mathbb{F}_t$ by computing $(\mathcal{B}_{H_t}, \mathcal{O}_{H_t}) := \text{FIND}\mathbf{H}(1^t)$. (See Parameter 3 in Definition 7.1.)
   (c) Find the polynomials $(N_{t,i}(x))_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)}$ that induce the "neighborhood" of each element in $\mathbb{F}_t$: for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, compute $[N_{t,i}]^{\mathsf{A}} := \text{FIND}\mathbf{N}(1^t, i)$. (See Parameter 5 in Definition 7.1.)
   (d) Find the constraint polynomial by computing $[P_t]^{\mathsf{A}} := \text{FIND}\mathbf{P}(1^t)$. (See Parameter 7 in Definition 7.1.)
   (e) Find the basis and offset for the $(\log|\mathtt{x}|)$-th affine subspace $I_{t,\log|\mathtt{x}|}$ in $\vec{I_t}$ by computing $(\mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}) := \text{FIND}\mathbf{I}(1^t, 1^{\log|\mathtt{x}|})$. (See Parameter 9 in Definition 7.1.)

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

   Invoke the (already "amplified") PCPP-verifier V_{aRS} for the Reed–Solomon code, with proximity parameter $\delta_{\text{RS}} := (8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$ 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

   $$V_{\text{aRS}}^{(p_0,\pi_0)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\text{RS}}, s'_{\text{RS}}\big)$$

   accepts.

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

   Invoke the (already "amplified") PCPP-verifier V_{aVRS} for the Vanishing Reed–Solomon code, with proximity parameter $\delta_{\text{VRS}} := 1/8$ and soundness $s'_{\text{VRS}} := 1/2$, on explicit input $(\mathbb{F}_t, \mathbb{F}_t, H_t, d)$, implicit input p_1, and proof of proximity π_1; that is, verify that

   $$V_{\text{aVRS}}^{(p_1,\pi_1)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{H_t}, \mathcal{O}_{H_t}, d, \delta_{\text{VRS}}, s'_{\text{VRS}}\big)$$

   accepts. Here $d := \deg(P_t(x, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \ldots, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathsf{c}_{\mathbf{N}}(t)})}))$.

4. *Consistency between* p_0 *and* p_1*:*
   (a) Draw a random $\alpha \in \mathbb{F}_t$.
   (b) For $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, compute $\alpha_i := [N_{t,i}]^{\mathsf{A}}(\alpha)$.
   (c) Query p_0 at the following points: $\alpha_1, \ldots, \alpha_{\mathsf{c}_{\mathbf{N}}(t)}$.
   (d) Query p_1 at the point α.
   (e) Compute $[P_t]^{\mathsf{A}} := \text{FIND}\mathbf{P}(1^t)$.
   (f) Compute $\omega := [P_t]^{\mathsf{A}}\big(\alpha, p_0(\alpha_1), \ldots, p_0(\alpha_{\mathsf{c}_{\mathbf{N}}(t)})\big)$.

(g) Verify that $p_1(\alpha) = \omega$.

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

   Define $A_{\mathtt{x}}$ to be the low-degree extension of the function $f_{\mathtt{x}} \colon I_{t,\log|\mathtt{x}|} \to \{0,1\}$ defined by $f_{\mathtt{x}}(\alpha_i) := \mathtt{x}_i$ where α_i is the i-th element in $I_{t,\log|\mathtt{x}|}$ and $I_{t,\log|\mathtt{x}|}$ is the $(\log|\mathtt{x}|)$-th affine subspace in $\vec{I}_t$; let $p_{\mathtt{x}}$ be the evaluation of $A_{\mathtt{x}}$ on $\mathbb{F}_t$.

   Invoke the (already "amplified") PCPP-verifier V_{aVRS} for the vanishing Reed–Solomon code, with proximity parameter $\delta_{\mathsf{C}} := (8\sum_{i=1}^{\mathbf{c_N}(t)} \deg(N_{t,i}))^{-1}$ and soundness $s'_{\mathsf{C}} := 1/2$, on explicit input $(\mathbb{F}_t, \mathbb{F}_t, I_{t,\log|\mathtt{x}|}, 2^{\mathbf{m_H}(t)} - 1)$, implicit input $p_0 - p_{\mathtt{x}}$, and proof of proximity π_{C}; that is, verify that

$$V_{\text{aVRS}}^{(p_0 - p_{\mathtt{x}}, \pi_{\mathsf{C}})}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}, 2^{\mathbf{m_H}(t)} - 1, \delta_{\mathsf{C}}, s'_{\mathsf{C}}\big)$$

   accepts.

Now we prove the correctness of the above constructions:

Proof of Theorem 8.1. We prove that the PCP verifier V_{sACSP} from Construction 8.5 has perfect completeness and soundness $1/2$, and, moreover, that the PCP prover P_{sACSP} from Construction 8.3, when provided with a valid witness, fulfills the perfect completeness of V_{sACSP}. The analysis of the query complexity, randomness complexity, and proof length complexity of the PCP system $(P_{\text{sACSP}}, V_{\text{sACSP}})$ is given in Appendix C (see Lemma C.8).

Completeness. We begin by proving that V_{sACSP} has perfect completeness. So suppose that $(\mathtt{x}, 1^t) \in \text{sACSP}$, 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 must be less than $2^{\mathbf{m_H}(t)}$; hence, $p_0 \in \text{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathbf{m_H}(t)} - 1)$. Invoking the completeness property of the PCPP verifier V_{aRS}, we deduce that there exists a string π_0 that makes $V_{\text{aRS}}^{(p_0,\pi_0)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, 2^{\mathbf{m_H}(t)} - 1, \delta_{\text{RS}}, s'_{\text{RS}}\big)$ accept with probability 1. We conclude that the same string π_0 makes the first subtest of V_{sACSP} accept with probability 1, because the first subtest of V_{sACSP} is the test $V_{\text{aRS}}^{(p_0,\pi_0)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, 2^{\mathbf{m_H}(t)} - 1, \delta_{\text{RS}}, s'_{\text{RS}}\big)$. (See Step 2 of V_{sACSP}.)

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:

$$B(x) := P_t\Big(x, A\big(N_{t,1}(x)\big), \dots, A\big(N_{t,\mathbf{c_N}(t)}(x)\big)\Big) \ .$$

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

$$\deg(B) \le d := \deg(P_t(x, x^{(2^{\mathbf{m_H}(t)}-1)\cdot\deg(N_{t,1})}, \dots, x^{(2^{\mathbf{m_H}(t)}-1)\cdot\deg(N_{t,\mathbf{c_N}(t)})})) \ ;$$

furthermore, because A is a witness for $(\mathtt{x}, 1^t) \in \text{sACSP}$, due to Item (i) in Definition 7.1, we know that B vanishes on the affine subspace H_t; hence, $p_1 \in \text{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d)$. Invoking the completeness property of the PCPP verifier V_{aVRS}, we deduce that there exists a string π_1 that makes $V_{\text{aVRS}}^{(p_1,\pi_1)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{H_t}, \mathcal{O}_{H_t}, d, \delta_{\text{VRS}}, s'_{\text{VRS}}\big)$ accept with probability 1. We conclude that the same string π_1 makes the second subtest of V_{sACSP} accept with probability 1, because the second subtest of V_{sACSP} is the test $V_{\text{aVRS}}^{(p_1,\pi_1)}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{H_t}, \mathcal{O}_{H_t}, d, \delta_{\text{VRS}}, s'_{\text{VRS}}\big)$. (See Step 3 of V_{sACSP}.)

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

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

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

Finally, because A is a witness for $(\mathtt{x}, 1^t) \in$ sACSP, due to Item (ii) in Definition 7.1, we know that A is consistent with the instance $(\mathtt{x}, 1^t)$, i.e., we know that $A(\alpha_i) = \mathtt{x}_i$ for $i = 1, \ldots, |\mathtt{x}|$, where α_i is the i-th element of $I_{t,\log|\mathtt{x}|}$ and $I_{t,\log|\mathtt{x}|}$ is the $(\log|\mathtt{x}|)$-th affine subspace in $\vec{I}_t$. Hence, letting $A_\mathtt{x}$ be the low-degree extension of the function $f_\mathtt{x} \colon I_{t,\log|\mathtt{x}|} \to \{0,1\}$ defined by $f_\mathtt{x}(\alpha_i) := \mathtt{x}_i$ where α_i is the i-th element in $I_{t,\log|\mathtt{x}|}$, and $p_\mathtt{x}$ the evaluation of $A_\mathtt{x}$ on $\mathbb{F}_t$, we know that $p_0 - p_\mathtt{x}$ is in $\text{VRS}(\mathbb{F}_t, \mathbb{F}_t, I_{t,\log|\mathtt{x}|}, 2^{\mathtt{m_H}(t)} - 1)$. Once again invoking the completeness property of the PCPP verifier V_{aVRS}, we deduce that there exists a string $\pi_\mathtt{c}$ that makes $V_{\text{aVRS}}^{(p_0 - p_\mathtt{x}, \pi_\mathtt{c})}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}, 2^{\mathtt{m_H}(t)} - 1, \delta_\mathtt{c}, s'_\mathtt{c}\big)$ accept with probability 1. We conclude that the same string $\pi_\mathtt{c}$ makes the fourth subtest of V_{sACSP} accept with probability 1, because the fourth (and last) subtest of V_{sACSP} is simply the test $V_{\text{aVRS}}^{(p_0 - p_\mathtt{x}, \pi_\mathtt{c})}\big(\mathcal{B}_{\mathbb{F}_t}, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}, 2^{\mathtt{m_H}(t)} - 1, \delta_\mathtt{c}, s'_\mathtt{c}\big)$. (See Step 5 of V_{sACSP}.)

Thus, we have established that V_{sACSP} has perfect completeness.

Finally, note that the PCP prover P_{sACSP}, on input $(\mathtt{x}, 1^t)$ and A, does indeed construct a proof $\pi = (p_0, \pi_1, p_1, \pi_1, \pi_\mathtt{c})$ that makes V_{sACSP} 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_{sACSP} has soundness $1/2$. As before, let $I_{t,\log|\mathtt{x}|}$ be the $(\log|\mathtt{x}|)$-th affine subspace in $\vec{I}_t$, $A_\mathtt{x}$ be the low-degree extension of the function $f_\mathtt{x} \colon I_{t,\log|\mathtt{x}|} \to \{0,1\}$ defined by $f_\mathtt{x}(\alpha_i) := \mathtt{x}_i$ where α_i is the i-th element in $I_{t,\log|\mathtt{x}|}$, and $p_\mathtt{x}$ the evaluation of $A_\mathtt{x}$ on $\mathbb{F}_t$. Also let $d := \deg(P_t(x, x^{(2^{\mathtt{m_H}(t)}-1)\cdot\deg(N_{t,1})}, \ldots, x^{(2^{\mathtt{m_H}(t)}-1)\cdot\deg(N_{t,\mathtt{c_N}(t)})}))$.

Suppose that $(\mathtt{x}, 1^t) \notin$ sACSP. We distinguish between four cases:

- *Case 1:* the function p_0 is $(8\sum_{i=1}^{\mathtt{c_N}(t)} \deg(N_{t,i}))^{-1}$-far from $\text{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathtt{m_H}(t)} - 1)$.

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

- *Case 2:* the function p_1 is 1/8-far from $\text{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d)$.

  Recall that the second subtest of V_{sACSP} (Step 3) tests proximity of p_1 to $\text{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d)$ 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_{sACSP} rejects with probability at least $1/2$.

- *Case 3:* the function $p_0 - p_\mathtt{x}$ is $(8\sum_{i=1}^{\mathtt{c_N}(t)} \deg(N_{t,i}))^{-1}$-far from $\text{VRS}(\mathbb{F}_t, \mathbb{F}_t, I_{t,\log|\mathtt{x}|}, 2^{\mathtt{m_H}(t)} - 1)$.

  Recall that the fourth subtest of V_{sACSP} (Step 5) tests proximity of $p_0 - p_\mathtt{x}$ to $\text{VRS}(\mathbb{F}_t, \mathbb{F}_t, I_{t,\log|\mathtt{x}|}, 2^{\mathtt{m_H}(t)} - 1)$ using the (amplified) VRS verifier V_{aVRS} with proximity parameter

$\delta_{\mathtt{c}} = (8\sum_{i=1}^{\mathtt{c_N}(t)} \deg(N_{t,i}))^{-1}$ and soundness $s'_{\mathtt{c}} = 1/2$; thus, in this case the second subtest of V_{sACSP} rejects with probability at least $1/2$.

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

  Let $A\colon \mathbb{F}_t \to \mathbb{F}_t$ be a polynomial of degree less than $2^{\mathtt{m_H}(t)}$ 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 $B(x) := P_t\big(x, A(N_{t,1}(x)), \ldots, A(N_{t,\mathtt{c_N}(t)}(x))\big)$. Note that A is unique, because $1-(2^{\mathtt{m_H}(t)}-1)/|\mathbb{F}_t|$ is larger than $2\delta_{\text{RS}} = (4\sum_{i=1}^{\mathtt{c_N}(t)} \deg(N_{t,i}))^{-1}$. (See Lemma 4.11.)

  Observe that $p_0 - p_{\mathtt{x}}$ is $(8\sum_{i=1}^{\mathtt{c_N}(t)} \deg(N_{t,i}))^{-1}$-close to the evaluation table of $A - A_{\mathtt{x}}$ over $\mathbb{F}_t$. Let $A'\colon \mathbb{F}_t \to \mathbb{F}_t$ be a polynomial of degree less than $2^{\mathtt{m_H}(t)}$ vanishing on $I_{t,\log|\mathtt{x}|}$ whose evaluation table over $\mathbb{F}_t$ is closest to $p_0 - p_{\mathtt{x}}$. Note that $A' = A - A_{\mathtt{x}}$ because both A' and $A - A_{\mathtt{x}}$ are polynomials of degree less than $2^{\mathtt{m_H}(t)}$ that are $(8\sum_{i=1}^{\mathtt{c_N}(t)} \deg(N_{t,i}))^{-1}$-close to $p_0 - p_{\mathtt{x}}$ and $1-(2^{\mathtt{m_H}(t)}-1)/|\mathbb{F}_t|$ is larger than $(4\sum_{i=1}^{\mathtt{c_N}(t)} \deg(N_{t,i}))^{-1}$. (See Lemma 4.11.) Thus we deduce that A is consistent with the instance $(\mathtt{x}, 1^t)$, i.e., satisfies Item (ii) in Definition 7.1.

  Because p_0 is $(8\sum_{i=1}^{\mathtt{c_N}(t)} \deg(N_{t,i}))^{-1}$-close to the evaluation table of A over $\mathbb{F}_t$, we deduce that, for $i = 1, \ldots, \mathtt{c_N}(t)$, $p_0 \circ N_{t,i}$ is $\frac{\deg(N_{t,i})}{8\sum_{i=1}^{\mathtt{c_N}(t)} \deg(N_{t,i})}$-close to the evaluation table of $A \circ N_{t,i}$.

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

  $$p_2(x) := P_t\Big(x, p_0\big(N_{t,1}(x)\big), \ldots, p_0\big(N_{t,\mathtt{c_N}(t)}(x)\big)\Big) \ .$$

  We deduce then, via a union bound, 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 vanishing on H_t whose evaluation table over $\mathbb{F}_t$ is closest to p_1. Again note that B' is unique, because $1 - d/|\mathbb{F}_t|$ is larger than $2\delta_{\text{VRS}} = 1/4$. (See Lemma 4.11.) 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 $(\mathtt{x}, 1^t)$) that $(\mathtt{x}, 1^t) \in \text{sACSP}$ — a contradiction. Thus, since both B and B' are (distinct) polynomials of degree at most $d \le |\mathbb{F}_t|/4$, 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_{sACSP} (Step 4) accepts with probability at most $1/8 + 1/8 + 1/4 = 1/2$.

This completes the proof of the soundness property. □

Remark 8.6. More generally, the soundness analysis would have also worked for any proximity parameter δ_{RS} for V_{aRS} and proximity parameters δ_{VRS} and $\delta_{\mathtt{c}}$ for V_{aVRS} as long as the following conditions hold: letting $d := \deg(P_t(x, x^{(2^{\mathtt{m_H}(t)}-1)\cdot\deg(N_{t,1})}, \ldots, x^{(2^{\mathtt{m_H}(t)}-1)\cdot\deg(N_{t,\mathtt{c_N}(t)})}))$,

$$1 - \frac{2^{\mathtt{m_H}(t)}-1}{|\mathbb{F}_t|} > 2\delta_{\text{RS}} \ , \qquad \text{(to ensure uniqueness of } A\text{)}$$

$$1 - \frac{2^{\mathtt{m_H}(t)}-1}{|\mathbb{F}_t|} > \delta_{\text{RS}} + \delta_{\mathtt{c}} \ , \qquad \text{(to ensure uniqueness of } A' \text{ and } A' = A - A_{\mathtt{x}}\text{)}$$

$$1 - \frac{d}{|\mathbb{F}_t|} > 2\delta_{\text{VRS}} \ , \qquad \text{(to ensure uniqueness of } B'\text{)}$$

$$\left(\sum_{i=1}^{\mathsf{c_N}(t)} \deg(N_{t,i})\right) \delta_{\mathrm{RS}} + \delta_{\mathrm{VRS}} + \frac{d}{|\mathbb{F}_t|} \leq \frac{1}{2} \ .$$

For example, for the specific choices of

$$\delta_{\mathrm{RS}} = \delta_{\mathsf{c}} = \frac{1}{8 \sum_{i=1}^{\mathsf{c_N}(t)} \deg(N_{t,i})} \quad \text{and} \quad \delta_{\mathrm{VRS}} = \frac{1}{8}$$

that we used in the proof of Theorem 8.1, all of the above constraints are implied by the sACSP algebraic constraint from Equation 3 of Definition 7.1.

Remark 8.7. The consistency with the instance requirement in the definition of sACSP (i.e., Item (ii) of Definition 7.1) is quite "special": it specifically requires that the concatenation of the vector of field elements $(A(\alpha_1), \dots, A(\alpha_{|\mathtt{x}|}))$ is, when properly converted to bits, equal to $\mathtt{x}$. This specific form of instance consistency enables us to use a simple test for verifying this requirement: namely, we can use a test of proximity to the appropriate Vanishing Reed–Solomon code. (See Step 5 of V_{sACSP}.)

It is possible to extend our construction to support more general forms of instance consistency where, for example, we are given some predicate that takes as input both $\mathtt{x}$ and $(A(\alpha_1), \dots, A(\alpha_{|\mathtt{x}|}))$ and decides whether the two are "consistent" or not (via a criterion that is perhaps different than mere equality between the two). To support any such a predicate, we would need to (1) account for the complexity of the predicate when viewed as a polynomial when specifying constraints on the field size, and (2) perform tests that are more complicated than simply testing proximity to an appropriate Vanishing Reed–Solomon code and are instead similar to how we verify that the constraint polynomial P_t is satisfied. (However, such an extension would complicate the PCP construction without much justification; see Footnote 16.)

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

For suppose that the instance $(\mathtt{x}, 1^t)$ is such that $\mathtt{x}$ 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 $|\mathtt{x}|$, but not during the online phase. In such a case PCPPs could help. However, we believe that there is a solution that (only comes at the expense of assuming the existence of one-way functions and) is more convenient in many settings: one can invoke the *Untrusted Input Lemma* of Ben-Sasson et al. [BSCGT13].[19]

Indeed, PCPs of Proximity have the disadvantage that one must have oracle access to the long input $\mathtt{x}$. Depending on the application, this may be quite inconvenient (especially if one wishes to conduct several computations on $\mathtt{x}$). On the other hand, the reduction in [BSCGT13] allows one to "move the input $\mathtt{x}$ to the witness" and replace $\mathtt{x}$ with a much shorter digest of it (which can be reused across multiple computations). Moreover, from an efficiency standpoint, enhancing our construction to a PCP of Proximity would make it somewhat less efficient, whereas the reduction in [BSCGT13] is much lighter.

So from both a convenience and efficiency perspective, a reduction like that in [BSCGT13] seems to be preferable for the purposes of handling long inputs.

[19] Such an invocation requires the PCP to satisfy a knowledge property; we prove that this is the case for our PCPs in the proof of our Theorem 4.

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 proof-length complexity and polylogarithmic randomness and query complexity. Our construction roughly follows their approach, but necessitates additional work for the PCP system to handle the more general language sACSP, to ensure that the verifier is succinct, and to ensure that the prover runs in quasilinear-time (and, moreover, that both verifier and prover run quasilinearly relative to the instance size).

The construction of this section implies as a special case a proof of a theorem left implicit in [BSGH$^+$05], where a special case of the language sACSP was considered for the purpose of constructing polylogarithmic-time verifiers for the PCPs of Ben-Sasson and Sudan. As discussed before, 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 prover and verifier in terms of the input instance (which we will also ensure is quasilinear). These concerns were not addressed before, and are ultimately crucial to prove our Theorem 1.

Remark 8.10 (combinatorial techniques)**.** While the early results on 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 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 seem 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 — such extensive algorithmic knowledge seems crucial for practical implementations of PCPs.

9 Proof of Theorem 3

In this section we prove Theorem 3, discussed in Section 1.5 and formally stated in Section 2.4. The theorem states the existence of a PCPP system (see Definition 2.5) for the code ensemble $\mathcal{E}_{\mathrm{RS}}$ (see Definition 2.9) having a concrete-efficiency threshold $\hat{B}_{\mathrm{RS}}(\mathbf{C}_{\times}) \leq 2^{43}$ (see Definition 2.7).

Recall that our proof strategy is in two steps; we describe these two steps in Section 9.1 and Section 9.2 respectively. We then concisely summarize the proof of the theorem in Section 9.3. Throughout, it will be useful to keep in mind the definitions from Section 4. Furthermore, because we are (for convenience) concentrating on $\hat{B}_{\mathrm{RS}}(\mathbf{C}_{\times})$, we shall simply write $\hat{B}_{\mathrm{RS}}$.

9.1 Step 1: Improving the Universal Constant of Bivariate Testing

The first step 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" forms the basis 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 9.1. *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*

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

Recall that $\delta_{A\times B}^{(d,*)}(f)$, $\delta_{A\times B}^{(*,e)}(f)$, and $\delta_{A\times B}^{(d,e)}(f)$ respectively denote the (fractional) 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 4.9.)

In other words, Lemma 9.1 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) (see Definition 4.6), 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 9.1 that ultimately allows for the construction of quasilinear-size proximity proofs in [BSS08] (via a recursive construction) 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, e.g., the corresponding theorem in [AS98] where the size of the domain had to quadratically depend on the degree.

Another aspect of Lemma 9.1, 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 Section 9.2.) 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 lemma in a more general form, letting c_0 depend on the fractional degree of each coordinate.

Theorem 9.2 (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

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

By setting $\frac{d}{m} := \frac{1}{4}$ and $\frac{e}{n} := \frac{1}{8}$ as in Lemma 9.1, 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 10.

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 9.1. Different settings of $\frac{d}{m}$ and $\frac{e}{n}$, eventually correspond (as can be seen by inspecting how Theorem 9.2 is eventually invoked in the soundness analysis in Section 11) 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 a construction with a good concrete-efficiency threshold.

We believe that Theorem 9.2 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.

9.2 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 3 is to provide a class of constructions that generalize the basic verifiers of [BSS08] for testing proximity to the Reed–Solomon code, 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 generalizations of the (strong) PCPP verifiers (see Definition 4.15) testing proximity to RS and VRS (see Definition 4.3 and Definition 4.4) constructed by Ben-Sasson and Sudan [BSS08] based on Lemma 9.1; 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 9.2 that strengthens and generalizes Lemma 9.1, as discussed in Section 9.1), 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 11.1 and proved in Section 11.

The focus of our effort is improving the soundness of $V_{\mathrm{RS},=}$, which is the PCPP verifier responsible for testing proximity to the code ensemble $\mathcal{E}_{\mathrm{RS}}$ (see Definition 2.9). 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 (roughly) greater than, less than, or equal to $1/8$ respectively; 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. See Figure 1; as discussed, $V_{\mathrm{RS},=}$ is "at the bottom of the stack".

It is for this reason that we have chosen to state Theorem 3 for the code ensemble $\mathcal{E}_{\mathrm{RS}}$, and thus henceforth our discussion will concentrate on $V_{\mathrm{RS},=}$. The relevant lemma for $V_{\mathrm{RS},=}$ from [BSS08] (stated for affine rather than linear subspaces) is:

Lemma 9.3. *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 affine 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, affine subspace) L. (See Definition 4.7.) 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. (Of course, the explicit input is suitably represented: $\mathbb{F}_{2^\ell}$ is represented via an irreducible polynomial and L via a basis and offset.)

Lemma 9.3 says that the (strong) PCPP verifier $V_{\mathrm{RS},=}$ has a soundness function $s_{\mathrm{RS},=}$ (see Definition 4.15) that can be lower bounded as follows:

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

Roughly, this means that $V_{\mathrm{RS},=}$ needs $\kappa^{\log c}/\delta$ sequential repetitions to have constant soundness (with proximity parameter δ). We have fixed $\delta = 1/3$ as a convention. (See Definition 2.5.)

Note that one cannot hope in an analysis showing that $s(O(1), n) \geq \kappa^{-o(1)}$, because $V_{\mathrm{RS},=}$ in [BSS08] and in our case consists of $O(\log\kappa)$ recursive invocations of a constant-soundness low-degree test (namely, the "random row or column test" mentioned in Section 9.1), so that we indeed expect $s(O(1), n) \leq \kappa^{-O(1)}$.

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 9.3 holds.*

The Ben-Sasson–Sudan Analysis. Ben-Sasson and Sudan proved Lemma 9.3 for

$$c = \big(64(c_0 + 2/3)\big)^{\frac{1}{\log(8/7)}} \quad [20]$$

and $c_0 = 128$. (As mentioned, the universal constant of bivariate testing c_0 affects the soundness analysis of $V_{\mathrm{RS},=}$). This yields $\log c \approx 67.5$, which is a very large value. Indeed, with this analysis, one can only show that $\hat{B}_{\mathrm{RS}}(\mathbf{C}_\times) \leq 2^{683}$. (Since values of c are quite large, we shall always give its *binary logarithm* when computing bounds for it.)

Even when invoking our Theorem 9.2 to obtain the smaller value $c_0 = 10.24$ to use in their soundness analysis, we get $\log c \approx 49.0$ — still very large.

Our Asymptotic Analysis. Without modifying the construction of [BSS08], we give an improved soundness analysis of $V_{\mathrm{RS},=}$ showing that we can take

$$c = \left(\frac{16}{3}(1+c_0) + 8 + 2\sqrt{16 + \frac{64}{3}(1+c_0)}\right)^{\frac{1}{\log(8/7)}}$$

so that, using $c_0 = 10.24$, we get $\log c \approx 34.5$. Unfortunately, $\log c \approx 34.5$ is still very large.

To tackle this problem, we develop an analysis that lets us efficiently explore a *class* of constructions for $V_{\mathrm{RS},=}$ related to the one in [BSS08]. Concretely, we parametrize several quantities in the construction of $V_{\mathrm{RS},=}$ (namely, the fractional degree to be tested, the query complexity, and how the recursion in $V_{\mathrm{RS},=}$ is "balanced") and *we are able to deduce an explicit expression for the constant c in terms of all of these parameters.* We are then in a position to identify better tradeoffs among these parameters.

Concretely, see Theorem 11.3 (and Corollary 11.4) in Section 11.1 for the formal statement giving the expression for c in terms of four parameters $(\eta, \kappa_0, \gamma, \mu)$, where $2^{-\eta}$ (roughly) denotes the fractional degree to be tested, 2^{κ_0} the number of queries of $V_{\mathrm{RS},=}$, γ how the affine subspace L is split in the recursion of $V_{\mathrm{RS},=}$, and μ what is the "overlap" between the two subspaces of L in the recursion of $V_{\mathrm{RS},=}$. In [BSS08], $\eta = 3$, $\kappa_0 = 6$, $\gamma = 0$, $\mu = 2$. In contrast, we can choose *any* setting of $(\eta, \kappa_0, \gamma, \mu)$ satisfying a certain set of conditions (see Equation 6). (In particular, our improved analysis also enables us to say something about the case $\eta = 2$, whereas previously the analysis was not strong enough to show any soundness for this setting; when $\eta = 2$, the fractional degree is roughly $1/4$ and yields a proximity proof shorter than tat in [BSS08].)

As an example, in Table 1, we give values for $\log c$ as we change η and κ_0 (and using an optimal choice of γ and μ for each choice of η and κ_0).

In particular, we are able to obtain $\log c \approx 16.0$ for a different construction than that in [BSS08] (that has the same proof length and number of queries). Unfortunately, $\log c \approx 16.0$ is still very large, and only lets us show $\hat{B}_{\mathrm{RS}}(\mathbf{C}_\times) \leq 2^{156}$.

[20] Note that [BSS08, Equation 15] is incorrect when $\kappa = 8$, so that, in [BSS08], they should have chosen c_1 to equal $\hat{c}^{\log(8/7)}/2$ instead of $\hat{c}^{\log(7/6)}/2$; thus, the exponent in the expression for c should indeed be $\frac{1}{\log(8/7)}$ instead of $\frac{1}{\log(7/6)}$.

$\log c$	$\kappa_0 = 4$	$\kappa_0 = 5$	$\kappa_0 = 6$	$\kappa_0 = 7$	$\kappa_0 = 8$	$\kappa_0 = 9$	$\kappa_0 = 10$
$\eta = 2$	29.0	29.0	19.0	19.0	16.0	16.0	14.0
$\eta = 3$	25.0	25.0	**16.0**	16.0	13.0	13.0	11.0
$\eta = 4$	n/a	n/a	29.0	29.0	18.0	18.0	14.0
$\eta = 5$	n/a	n/a	35.0	35.0	21.0	21.0	16.0

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 $\log c \approx 16.0$ for the setting $\eta = 3$ and $\kappa_0 = 6$ (in bold) corresponds to what we obtain for (a fine tuning of) the construction in [BSS08], improving on the original $\log c \approx 67.5$. (Cells with "n/a" correspond to parameter choices for which the construction is not defined.)

Our Instance-Size Specific Analysis. We do not know how to push further our asymptotic analysis mentioned in the previous paragraph. To go beyond it, we depart from the "standard" asymptotic perspective, and instead seek to perform an analysis that is tailored to the specific instance size, i.e., tailored to the specific dimension κ of the affine subspace L.

Starting from our asymptotic analysis, we show how to derive an efficiently-computable recursive function $D(\cdot)$ that lets us *numerically compute* a lower bound on the soundness for any given problem size: namely, $s(\delta, \kappa) \geq \delta / D(\kappa)$ holds for all dimensions κ. See Section 11.1.4 (specifically, Equation 25) for our result.

We discover that $D(\kappa)$ approaches *very slowly* our asymptotic upper bound of $\kappa^{\log c}$. For instance, we can define

$$\log \tilde{c} \stackrel{\text{def}}{=} \max_{k \in \{1,\ldots,2^{100}\}} \frac{\log D(\kappa)}{\log \kappa} \quad ,$$

to be the "effective" $\log c$ for instances of size up to 2^{100}. As an example, in Table 2, we give values for $\log \tilde{c}$ for the same choices of parameters that we used in Table 1; we see great improvements throughout the table. For instance, we obtain that $\log \tilde{c} \leq 5.8$, which is much less than $\log c \approx 16.0$ obtained in the asymptotic analysis.

$\log \tilde{c}$	$\kappa_0 = 4$	$\kappa_0 = 5$	$\kappa_0 = 6$	$\kappa_0 = 7$	$\kappa_0 = 8$	$\kappa_0 = 9$	$\kappa_0 = 10$
$\eta = 2$	9.0	8.1	7.6	7.3	6.8	6.4	6.3
$\eta = 3$	8.0	6.8	**5.8**	5.3	5.1	4.8	4.5
$\eta = 4$	n/a	n/a	5.8	5.3	5.0	4.6	4.1
$\eta = 5$	n/a	n/a	7.4	6.3	5.1	4.9	4.6

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 asymptotic value 16.0 to the effective value $\log \tilde{c} \approx 5.8$. (Cells with "n/a" correspond to parameter choices for which the construction is not defined.)

9.3 Step 3: Putting Things Together

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

$$\mathcal{E}^{(\eta)} \stackrel{\text{def}}{=} \left\{ \mathrm{RS}(\mathbb{F}_{2^\ell}, L, d) \;\middle|\; L \text{ is an affine subspace of } \mathbb{F}_{2^\ell}\,,\ d = 2^{-\eta}|L| - 1 \right\} \ . \tag{4}$$

Any code $C \in \mathcal{E}_{\mathrm{RS}}$ is thus fully specified by ℓ, L, and d. With Definition 2.7 in mind, we can define

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

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 (specifically, as mentioned, we use a proximity parameter of $1/3$).

Recall that, in Theorem 3, we consider the specific case of testing proximity to the code ensemble $\mathcal{E}_{\mathrm{RS}} = \mathcal{E}^{(3)}$ (see Definition 2.9) and upper bounding the concrete-efficiency threshold for the cost function $\mathbf{C}_\times(a, b) := a \cdot b$ (i.e., upper bounding $\hat{B}^{(3)}(\mathbf{C}_\times)$). As mentioned, these choices are only for concreteness, because our analysis lets us upper bound $\hat{B}^{(\eta)}(\mathbf{C})$ for any η and any polynomial cost cost function $\mathbf{C}$ (in fact, any efficiently-computable cost function too).

Therefore, we now explain how the two steps described in the previous two subsections come together to give a proof of Theorem 3, while keeping most of the proof in the general case.

Proof of Theorem 3. Recall from Definition 2.7 that, in order to compute $\hat{B}^{(\eta)}(\mathbf{C})$ for a given PCPP system testing proximity to $\mathcal{E}^{(\eta)}$, we need to be able to compute (or at least upper bound), for any code $C \in \mathcal{E}^{(\eta)}$, both $L(C)$ and $Q(C)$.

As discussed, we have developed a *class* of proximity testers $\mathcal{E}^{(\eta)}$. For the member of this class corresponding to a choice $(\eta, \kappa_0, \gamma, \mu)$ of parameters, and for each $C \in \mathcal{E}^{(\eta)}$ with block length 2^κ:

- we can either use the bound $L(C) \leq 2^\kappa \cdot \kappa^{1+\max\{-\gamma+\mu+1, \gamma\}}$ (see Lemma C.1) or numerically compute $L(C)$ (see Lemma C.10);
- we can either use the bound $Q(C) \leq 2^{\kappa_0} \cdot 3\kappa^{\log c(\eta,\kappa_0,\gamma,\mu)}$, where $\log c(\eta, \kappa_0, \gamma, \mu)$ is the asymptotic value from Theorem 11.3, or rely on our instance-size specific analysis to numerically compute $D(\kappa)$ (see Equation 25) such that $Q(C) \leq 2^{\kappa_0} \cdot 3D(\kappa)$. (Note that the number of queries 2^{κ_0} of $V_{\mathrm{RS},=}$ has been multiplied to take into account the fact that we need to amplify soundness to $1/2$ and have proximity parameter $1/3$; see Remark 11.2.)

Either way, we obtain some bound $L_{(\eta,\kappa_0,\gamma,\mu)}$ for $L(C)$ and some bound $Q_{(\eta,\kappa_0,\gamma,\mu)}$ for $Q(C)$. Recalling that $n = |L| = 2^\kappa$ and $k = 2^{-\eta}|L| = 2^{\kappa-\eta}$, if

$$\mathbf{C}\left(\frac{2^\kappa + L_{(\eta,\kappa_0,\gamma,\mu)}}{2^{\kappa-\eta}}\,,\, Q_{(\eta,\kappa_0,\gamma,\mu)}\right) < 2^{\kappa-\eta}$$

then

$$\mathbf{C}\left(\frac{n + L(C)}{k}\,,\, Q(C)\right) < k \ .$$

We can then compute a bound for $\hat{B}^{(\eta)}(\mathbf{C})$ for any choice of $(\eta, \kappa_0, \gamma, \mu)$.

For example, using the parameters $(\eta, \kappa_0, \gamma, \mu) = (3, 15, 1, 2)$, we get $\hat{B}_{\mathrm{RS}} = \hat{B}^{(3)}(\mathbf{C}_\times) \leq 2^{43}$, as claimed. (In contrast, using the original parameters $(\eta, \kappa_0, \gamma, \mu) = (3, 6, 0, 2)$ and the bound $\log c \approx 67.5$ obtained in [BSS08], we only get $\hat{B}_{\mathrm{RS}}(\mathbf{C}_\times) \leq 2^{683}$.) $\square$

See `http://people.csail.mit.edu/alexch/research/tppcp/` for a Mathematica notebook that contains most of the computations carried out in this section and enables one, given η and $\mathbf{C}$, to compute $\hat{B}^{(\eta)}(\mathbf{C})$ for both our construction (and, for comparison, the one in [BSS08]).

10 Improved Universal Constant of Bivariate Testing

After recalling some basic lemmas about polynomials from [Spi95] (Section 10.1), we prove Theorem 9.2 in two steps (respectively in Section 10.2 and Section 10.3); as an example, we then show how to obtain the improved universal constant of bivariate testing $c_0 = 10.24$ (Section 10.4).

10.1 Some Basic Lemmas

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

Lemma 10.1 ([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 10.2 ([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 10.3 ([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 10.4 ([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].) □

10.2 The PS Bivariate Testing Theorem

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

Theorem 10.5. *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 10.5. 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 10.2, 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 10.1, 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 10.3 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 10.4 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

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

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 $\deg_y E \leq \delta n$. □

10.2.1 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 10.5. So let us briefly discuss how this requirement arises in the proof of Theorem 10.5. 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 10.3) 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 10.4) 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 limit to Lemma 10.4. In the proof of Theorem 10.5 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 10.4.

It is easy to note, however, that the requirement $1 > \delta' + \varepsilon' + \delta + \varepsilon$ in Lemma 10.4 (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 10.5 to a requirement such as $2 > \frac{d}{m} + \frac{e}{n} + 2\delta$ would be great, and in fact seems a reasonable goal.

10.3 The Universal Constant for Bivariate Testing

We re-prove Lemma 9.1 via a 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 10.5 holds for the proximity parameter $\delta > 0$ and fractional degrees $\frac{d}{m}$ and $\frac{e}{n}$.[21] (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 10.6. *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 9.1. 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 10.6. 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:

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

and

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

[21] 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 10.5 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 $\delta_{A\times B}^{(d,e)}(g)$ derived earlier, we obtain

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

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

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

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

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

10.4 Putting Things Together

Combining Theorem 10.5 and Lemma 10.6, we obtain Theorem 9.2.

For example, by using the best predicate $\Phi(\delta, d, m, e, n)$ we could prove (namely, $1 \overset{?}{>} \frac{d}{m} + \frac{e}{n} + 2\delta$ as the predicate $\Phi(\delta, d, m, e, n)$ from Theorem 10.5) and by setting $\frac{d}{m} = 1/4$ and $\frac{e}{n} = 1/8$ as in Lemma 9.1, we recover the (improved) constant $c_0 = 10.24$, as opposed to the previous value $c_0 = 128$.

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

We prove a much tighter soundness analysis for the verifiers V_{RS} and V_{VRS} (see Algorithm 13 and Algorithm 12), which respectively are supposed to test proximity to RS and VRS over affine subspaces of finite field extensions of $\mathbb{F}_2$. We show the following theorem:

Theorem 11.1. *The (strong) PCPP verifiers V_{RS} and V_{VRS} have respective soundness functions*

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

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

$$\begin{gathered} \min\left\{ \left\lfloor \tfrac{\kappa_0+1}{2} \right\rfloor - \gamma \, , \, \left\lceil \tfrac{\kappa_0+1}{2} \right\rceil + \gamma \right\} \geq 1 \\ \max\left\{ \left\lfloor \tfrac{\kappa_0+1}{2} \right\rfloor - \gamma + \mu + 1 \, , \, \left\lceil \tfrac{\kappa_0+1}{2} \right\rceil + \gamma \right\} \leq \kappa_0 \\ \min\left\{ \left\lfloor \tfrac{\kappa_0+1}{2} \right\rfloor - \gamma + \mu + 1 \, , \, \left\lceil \tfrac{\kappa_0+1}{2} \right\rceil + \gamma \right\} \geq \eta + 1 \\ \kappa_0 + \frac{3+(-1)^{\kappa_0 \bmod 2}}{2} > \left\lfloor \frac{\kappa_0 + \frac{3+(-1)^{\kappa_0 \bmod 2}}{2}}{2} \right\rfloor - \gamma + \mu + 1 \quad , \\ \kappa_0 + \frac{1+(-1)^{\kappa_0 \bmod 2}}{2} > \left\lfloor \frac{\kappa_0 + \frac{1+(-1)^{\kappa_0 \bmod 2}}{2}}{2} \right\rfloor + \gamma \\ \mu + 1 \geq \eta \geq \max\{2, \mu + 1\} \\ \mu \geq \gamma \end{gathered}$$

and

$$c \stackrel{\text{def}}{=} \left(a + \frac{b}{2} + \frac{\sqrt{b(b+4a)}}{2} \right)^{\frac{1}{\min\{e_1, e_2\}}} \quad , \text{ where } \begin{cases} c_0 \stackrel{\text{def}}{=} \frac{4}{\left(1 - \frac{1}{2^{\eta-1}} - \frac{1}{2^\eta}\right)^2} \\ a \stackrel{\text{def}}{=} \dfrac{2^{\eta+1}(1+c_0)}{2^{\eta-1}-1} > 0 \\ b \stackrel{\text{def}}{=} 2^{\mu+2} > 0 \\ e_1 \stackrel{\text{def}}{=} \log\left(\frac{\kappa_0 + \frac{3+(-1)^{\kappa_0 \bmod 2}}{2}}{\lfloor (\kappa_0 + \frac{3+(-1)^{\kappa_0 \bmod 2}}{2})/2 \rfloor - \gamma + \mu + 1} \right) \\ e_2 \stackrel{\text{def}}{=} \log\left(\frac{\kappa_0 + \frac{1+(-1)^{\kappa_0 \bmod 2}}{2}}{\lceil (\kappa_0 + \frac{1+(-1)^{\kappa_0 \bmod 2}}{2})/2 \rceil + \gamma} \right) \end{cases} .$$

(Alternatively, for any given problem size $n = 2^\kappa$, one can use the method discussed in Section 11.1.4 to compute an "effective" c, which is usually much better than the expression given above that simultaneously works for all n.)

We perform our improved soundness analyses "from bottom to top" (see the lowermost five "verifier boxes" in Figure 1 for dependencies):

- in Section 11.1 we analyze the soundness of $V_{\mathrm{RS},=}$;
- in Section 11.2 we analyze the soundness of $V_{\mathrm{RS},<}$;
- in Section 11.3 we analyze the soundness of $V_{\mathrm{RS},>}$;
- in Section 11.4 we analyze the soundness of V_{RS}; and

- in Section 11.5 we analyze the soundness of V_{RS}.

Remark 11.2 (From Strong PCPPs to "Weak" PCPPs)**.** The PCPPs obtained in Theorem 11.1 are of the "strong" kind, i.e., follow Definition 4.15, instead of the weaker Definition 4.14.

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

$$m_{s,\delta,s'}(n) = \left\lceil \frac{\log(1-s')}{\log\left(1 - s(\delta,n)\right)} \right\rceil \ , \tag{5}$$

where n is the length of the explicit input.

We are not interested in randomness-efficient sequential repetition (see [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

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

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

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

and

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

In general, we denote by

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

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. See Algorithm 11 and Algorithm 10 for more details.

11.1 Soundness Analysis of $V_{\mathrm{RS},=}$

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

As mentioned in the introduction, we have generalized the construction of $V_{\mathrm{RS},=}$ (its code is given in Algorithm 16) and we shall provide a much tighter soundness analysis, provided by the following theorem:

Theorem 11.3. *Consider a construction of $V_{\mathrm{RS},=}$ parametrized by integers $(\eta, \kappa_0, \gamma, \mu)$ as described above, with*

$$\begin{gathered}
\min\left\{ \left\lfloor \tfrac{\kappa_0+1}{2} \right\rfloor - \gamma \,,\, \left\lceil \tfrac{\kappa_0+1}{2} \right\rceil + \gamma \right\} \geq 1 \\
\max\left\{ \left\lfloor \tfrac{\kappa_0+1}{2} \right\rfloor - \gamma + \mu + 1 \,,\, \left\lceil \tfrac{\kappa_0+1}{2} \right\rceil + \gamma \right\} \leq \kappa_0 \\
\min\left\{ \left\lfloor \tfrac{\kappa_0+1}{2} \right\rfloor - \gamma + \mu + 1 \,,\, \left\lceil \tfrac{\kappa_0+1}{2} \right\rceil + \gamma \right\} \geq \eta + 1 \\
\kappa_0 + \frac{3+(-1)^{\kappa_0 \bmod 2}}{2} > \left\lfloor \frac{\kappa_0 + \frac{3+(-1)^{\kappa_0 \bmod 2}}{2}}{2} \right\rfloor - \gamma + \mu + 1 \\
\kappa_0 + \frac{1+(-1)^{\kappa_0 \bmod 2}}{2} > \left\lceil \frac{\kappa_0 + \frac{1+(-1)^{\kappa_0 \bmod 2}}{2}}{2} \right\rceil + \gamma \\
\mu + 1 \geq \eta \geq \max\{2, \mu+1\} \\
\mu \geq \gamma
\end{gathered} \quad . \tag{6}$$

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 affine subspace $L \subseteq \mathbb{F}_{2^\ell}$ it holds that

$$\Pr\left[V_{\mathrm{RS},=}^{(p,\pi)}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big) = 1 \right] > 1 - \varepsilon \tag{7}$$

then

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

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

$$c \stackrel{\text{def}}{=} \left(a + \frac{b}{2} + \frac{\sqrt{b(b+4a)}}{2} \right)^{\frac{1}{\min\{e_1, e_2\}}} \ , \ where \ \begin{cases} c_0 \stackrel{\text{def}}{=} \dfrac{4}{\left(1 - \frac{1}{2^{\eta-1}} - \frac{1}{2^\eta}\right)^2} \\ a \stackrel{\text{def}}{=} \dfrac{2^{\eta+1}(1+c_0)}{2^{\eta-1}-1} > 0 \\ b \stackrel{\text{def}}{=} 2^{\mu+2} > 0 \\ e_1 \stackrel{\text{def}}{=} \log\left(\dfrac{\kappa_0 + \frac{3+(-1)^{\kappa_0 \bmod 2}}{2}}{\lfloor (\kappa_0 + \frac{3+(-1)^{\kappa_0 \bmod 2}}{2})/2 \rfloor - \gamma + \mu + 1} \right) \\ e_2 \stackrel{\text{def}}{=} \log\left(\dfrac{\kappa_0 + \frac{1+(-1)^{\kappa_0 \bmod 2}}{2}}{\lceil (\kappa_0 + \frac{1+(-1)^{\kappa_0 \bmod 2}}{2})/2 \rceil + \gamma} \right) \end{cases} .$$

(In particular, we are forced to choose $\mu = \eta - 1$ and we can optimally choose γ as the integer, among those allowed by Equation 6, minimizing $\frac{1}{\min\{e_1, e_2\}}$.)

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

Corollary 11.4. *The (strong) PCPP verifier $V_{\mathrm{RS},=}$ has soundness function*

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

We first prove some lemmas (Section 11.1.1) and introduce basic notation and facts for the proof (Section 11.1.2), then provide the asymptotic analysis of the soundness function (Section 11.1.3), and after that we explain how our analysis lets us numerically compute a better lower bound on the soundness for any given dimension (Section 11.1.4).

Throughout, 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 $g|_\beta^{\leftrightarrow}$ and the restriction of g to the α-column of W is denoted by $g|_\alpha^{\updownarrow}$.

11.1.1 Some lemmas

We begin by proving some lemmas needed for the soundness analysis of $V_{\mathrm{RS},=}$ in Section 11.1.3.

The first two 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 11.5. *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:*

$$\delta_{A\times B}^{(d,*)}(g) = \mathop{\mathbf{E}}_{b\in B}\left[\delta_A^{(d)}(g|_b^{\leftrightarrow})\right] \ , \tag{9}$$

where $g|_b^{\leftrightarrow} \colon A \to \mathbb{F}$ denotes the restriction of g to the b-th row.

Lemma 11.6. *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:*

$$\delta_{A\times B}^{(*,e)}(g) = \mathop{\mathbf{E}}_{a\in A}\left[\delta_B^{(e)}(g|_a^{\updownarrow})\right] \ , \tag{10}$$

where $g|_a^{\updownarrow} \colon B \to \mathbb{F}$ denotes the restriction of g to the a-th column.

The proofs of these lemmas were left implicit in [BSS08]. For completeness, we give the proof of Lemma 11.5; an analogous argument can be carried out to prove the (symmetric) Lemma 11.6.

Proof of Lemma 11.5. Re-writing the left-hand side of Equation 9:

$$\begin{aligned}
\delta_{A\times B}^{(d,*)}(g) &= \min_{\substack{Q\in\mathbb{F}[x,y]\\ \deg_x(Q)\le d}} \Delta_{A\times B}(g, Q) \\
&= \min_{\substack{Q\in\mathbb{F}[x,y]\\ \deg_x(Q)\le d}} \frac{\left|\left\{(a,b)\in A\times B \,:\, g(a,b)\neq Q(a,b)\right\}\right|}{|A|\cdot|B|} \\
&= \frac{1}{|A|\cdot|B|}\cdot \min_{\substack{Q\in\mathbb{F}[x,y]\\ \deg_x(Q)\le d}} \sum_{b\in B}\left|\{a\in A \,:\, g(a,b)\neq Q(a,b)\}\right| \ .
\end{aligned} \tag{11}$$

Re-writing the right-hand side of Equation 9:

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

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

Let P be a bivariate polynomial over $\mathbb{F}$ (with degree in x at most d) that minimizes the summation in Equation 11; hence,

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

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

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

Thus,

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

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 12, so that

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

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,

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

$$
\begin{aligned}
&\geq \frac{1}{|A|\cdot|B|}\cdot \min_{\substack{Q\in\mathbb{F}[x,y]\\ \deg_x(Q)\leq d}} \sum_{b\in B}\Big|\big\{a\in A \,:\, g(a,b)\neq Q(a,b)\big\}\Big| \\
&= \delta^{(d,*)}_{A\times B}(g) \ . \qquad (14)
\end{aligned}
$$

From Equation 13 and Equation 14, we deduce Equation 9, as desired. □

The next lemma highlights some properties of vanishing polynomials for affine subspaces, generalizing those presented in [BSS08, Proposition 6.4].

Lemma 11.7. *Let L be an affine subspace of $\mathbb{F}_{2^\ell}$ with a basis $\mathcal{B}_L = (a_1,\ldots,a_\kappa)$ and offset $\mathcal{O}_L$, and let $j\in\{1,\ldots,\kappa-1\}$. Define:*

$$
\begin{array}{lcl}
L_0^* := \mathrm{span}(a_1,\ldots,a_j) & \quad \textit{and} \quad & L_1^* := \mathrm{span}(a_{j+1},\ldots,a_\kappa) \\
L_0 := L_0^* + \mathcal{O}_L & & L_1 := L_1^* + \mathcal{O}_L
\end{array} \ .
$$

Let Z_{L_0} be the vanishing polynomial of L_0, i.e., $Z_{L_0}(x) := \prod_{\alpha\in L_0}(x-\alpha)$. Then:

- *The polynomial Z_{L_0} is an $\mathbb{F}_2$-affine map and its kernel is L_0.*
- *$Z_{L_0}(L) = Z_{L_0}(L_1)$ and $Z_{L_0}(L_1)$ is a linear subspace of dimension $\dim(L_1)$.*
- *Z_{L_0} is a one-to-one map from L_1 to $Z_{L_0}(L_1)$.*
- *Z_{L_0} is a $|L_0|$-to-one map from L to $Z_{L_0}(L_1)$; moreover the affine subspace $L_0+\beta+\mathcal{O}_L$ is mapped to $Z_{L_0}(\beta)$ for each $\beta\in L_1$.*

Proof. The above properties follow from observations on the form of Z_{L_0}; see Algorithm 21.

More precisely, $Z_{L_0}(x) = Z_{L_0^*}(x) + Z_{L_0^*}(\mathcal{O}_L)$, where $Z_{L_0^*}$ is the vanishing polynomial of L_0^*; moreover, $Z_{L_0^*}(x) = \sum_{i=0}^{j}\beta_i x^{2^i}$ for some coefficients $\beta_1,\ldots,\beta_j$, so that $Z_{L_0^*}$ is an $\mathbb{F}_2$-linear map with kernel equal to L_0^*. We deduce that Z_{L_0} is an $\mathbb{F}_2$-affine map and its kernel is L_0.

Next, since $L = (L_0^*\oplus L_1^*)+\mathcal{O}_L$, we know that

$$
\begin{aligned}
Z_{L_0}(L) &= Z_{L_0}\big((L_0^*\oplus L_1^*)+\mathcal{O}_L\big) \\
&= Z_{L_0^*}\big((L_0^*\oplus L_1^*)+\mathcal{O}_L\big) + Z_{L_0^*}(\mathcal{O}_L) \\
&= Z_{L_0^*}\big(L_1^*+\mathcal{O}_L\big) + Z_{L_0^*}(\mathcal{O}_L) \\
&= Z_{L_0}(L_1) \ ;
\end{aligned}
$$

moreover, the above derivation tells us that $Z_{L_0}(L_1) = Z_{L_0^*}(L_1^*)$, which is a linear subspace of dimension $\dim(L_1^*) = \dim(L_1)$.

Next, it is easy to show that $Z_{L_0^*}$ is a one-to-one map from L_1^* to $Z_{L_0^*}(L_1^*)$ and that $Z_{L_0^*}$ is a $|L_0^*|$-to-one map from $L_0^*\oplus L_1^*$ to $Z_{L_0^*}(L_1^*)$. From these facts we immediately deduce that Z_{L_0} is a one-to-one map from L_1 to $Z_{L_0}(L_1)$ and Z_{L_0} is a $|L_0|$-to-one map from L to $Z_{L_0}(L_1)$.

Finally, for $\beta\in L_1$,

$$
\begin{aligned}
Z_{L_0}(L_0+\beta+\mathcal{O}_L) &= Z_{L_0}\big((L_0^*+\mathcal{O}_L)+\beta+\mathcal{O}_L\big) \\
&= Z_{L_0^*}\big((L_0^*+\mathcal{O}_L)+\beta+\mathcal{O}_L\big) + Z_{L_0^*}(\mathcal{O}_L) \\
&= Z_{L_0^*}(\beta) + Z_{L_0^*}(\mathcal{O}_L) \\
&= Z_{L_0}(\beta) \ ,
\end{aligned}
$$

as desired. □

11.1.2 Basic notation

We introduce basic notation and facts needed for the soundness analysis of $V_{\text{RS},=}$ in Section 11.1.3.

Let $(a_1, \ldots, a_\kappa)$ be any basis for the given affine subspace L; let $\mathcal{O}$ be the offset of L. Define the affine subspaces L_0, L_0', L_1, L_β of L as follows:

$$L_0 := \operatorname{span}(a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma}) + \mathcal{O} \ \text{, so that } \dim(L_0) = \lfloor \tfrac{\kappa}{2} \rfloor - \gamma \ ;$$

$$L_0' := \operatorname{span}(a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}) + \mathcal{O} \ \text{, so that } \dim(L_0') = \lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu \ ;$$

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

$$L_1 := \operatorname{span}(a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_\kappa) + \mathcal{O} \ \text{, so that } \dim(L_1) = \lceil \tfrac{\kappa}{2} \rceil + \gamma \ .$$

Note that L_0, L_0', L_1, L_β are defined similarly to [BSS08, Definition 6.4], with the exception of the additional parameters γ and μ that we introduce to control their sizes and the fact that we work with affine (rather than linear) subspaces. 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 6 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").

Define the linearized polynomial Z_{L_0} to be the vanishing polynomial of L_0; more precisely, since Z_{L_0} is a vanishing polynomial of an affine (rather than linear) subspace, the polynomial Z_{L_0} is an affine (rather than linear) map. (See Lemma 11.7.)

Define the two disjoint subsets T and S of $\mathbb{F}_{2^\ell} \times \mathbb{F}_{2^\ell}$ as follows:

$$T := \bigcup_{\beta \in L_1} \Big((L_0 + \beta + \mathcal{O}_L) \times \{Z_{L_0}(\beta)\} \Big) \quad \text{and} \quad S := \left(\bigcup_{\beta \in L_1} \big(L_\beta \times \{Z_{L_0}(\beta)\} \big) \right) - T \ .$$

We now introduce the "format" of a proximity proof π expected by $V_{\text{RS},=}^{(p,\pi)}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big)$; this format is defined by induction on κ and generalizes [BSS08, Definition 6.4]. If $\kappa \leq \kappa_0$, then π is empty. Otherwise, $\pi = (f, \Pi)$ where f is a partial bivariate function over partial domain S and Π is a sequence of proximity proofs for Reed–Solomon codes over smaller affine subspaces. Concretely, Π has a subproof $\pi_{\beta'}^{\leftrightarrow}$ for a Reed–Solomon codeword over L_β for each $\beta' = Z_{L_0}(\beta) \in Z_{L_0}(L_1)$ and has a subproof $\pi_{\alpha}^{\updownarrow}$ for a Reed–Solomon codeword over $Z_{L_0}(L_1)$ for each $\alpha \in L_0'$; thus, overall, $\Pi = \{\pi_{\beta'}^{\leftrightarrow} \,:\, \beta' \in Z_{L_0}(L_1)\} \cup \{\pi_{\alpha}^{\updownarrow} \,:\, \alpha \in L_0'\}$.

The next lemma extends [BSS08, Proposition 6.5] to affine subspaces, showing that $S \cup T$ can be decomposed into rows and columns that are affine subspaces of size approximately $\sqrt{|L|}$.

Lemma 11.8. *The set $S \cup T$ is the disjoint union of $Z_{L_0}(\beta)$-rows taken over $\beta \in L_1$; moreover, the $Z_{L_0}(\beta)$-row of $S \cup T$ is the affine subspace L_β. Similarly, for every $\alpha \in L_0'$, the α-column of $S \cup T$ is the affine (in fact, linear) subspace $Z_{L_0}(L_1)$.*

Proof. Fix $\beta \in L_1$. By the last property of Lemma 11.7, the $Z_{L_0}(\beta)$-row of T is equal to $L_0 + \beta + \mathcal{O}_L$; by the definition of L_β, $L_0 + \beta + \mathcal{O}_L \subseteq L_\beta$. We deduce that $S \cup T = \cup_{\beta \in L_1} \big(L_\beta \times \{Z_{L_0}(\beta)\} \big)$.

In particular, $S \cup T$ is indeed a disjoint union of $Z_{L_0}(\beta)$-rows, where each $Z_{L_0}(\beta)$-row is equal to L_β.

Now fix $\alpha \in L_0'$. For every $\beta \in L_1$, we have $L_0' \subseteq L_\beta$ and $L_\beta \times \{Z_{L_0}(\beta)\} \subseteq S$ and thus $(\alpha, Z_{L_0}(\beta)) \in S$; hence, $Z_{L_0}(\beta)$ is a subset of the α-column. However, the only non-empty rows in $S \cup T$ are the $Z_{L_0}(\beta)$-rows, so that the α-column of $S \cup T$ is precisely $Z_{L_0}(L_1)$. $\square$

The proximity proof $\pi = (f, \Pi)$ as in [BSS08, Definition 6.4]. Finally, recall 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,

Given the above notation,

11.1.3 The proof by induction

We proceed to the proof of Theorem 11.3. 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 our generalized $V_{\mathrm{RS},=}$, see Algorithm 16; throughout, we assume basic familiarity with $V_{\mathrm{RS},=}$.

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 11.3. 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 7); 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 8), and want the smallest c such that $D(\kappa) \leq c^{\log \kappa}$.

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

- row test: choose a random $\beta \in L_1$ and recurse on $V_{\mathrm{RS},=}^{(\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)}, \pi^{\leftrightarrow}_{Z_{L_0}(\beta)})}\big(\mathbb{F}_{2^\ell}, L_\beta, |L_\beta|/2^\eta - 1\big)$, or
- column test: choose a random $\alpha \in L_0'$ and recurse on $V_{\mathrm{RS},=}^{(\hat{f}|^{\updownarrow}_{\alpha}, \pi^{\updownarrow}_{\alpha})}\big(\mathbb{F}_{2^\ell}, Z_{L_0}(L_1), |Z_{L_0}(L_1)|/2^\eta - 1\big)$.

Step 1: restricting the bivariate function $\hat{f}$ to a product set $L_0' \times Z_{L_0}(L_1)$. Define the following quantities:

- for every $\beta \in L_1$, $\varepsilon(\beta)$ is the probability that the inner verifier rejects $(\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)}, \pi^{\leftrightarrow}_{Z_{L_0}(\beta)})$;
- 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 $\beta \in L_1$);
- $e := |Z_{L_0}(L_1)|/2^\eta - 1$.

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, $\varepsilon_{\mathsf{row}}, \varepsilon_{\mathsf{col}} \le 2\varepsilon$. Also, by Lemma 11.8, we deduce that:

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

Using the inductive assumption, we deduce that:

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

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

- Using Lemma 11.5, as well as the facts that $L_0' \subseteq L_\beta$ and $|L_0'| = |L_\beta|/2$, we deduce that
$$\delta^{(d,*)}_{L_0' \times Z_{L_0}(L_1)}(f') = \mathop{\mathbf{E}}_{\beta \in L_1}\left[\delta^{(d)}_{L_0'}(f'|^{\leftrightarrow}_{Z_{L_0}(\beta)})\right] \le \mathop{\mathbf{E}}_{\beta \in L_1}\left[2\cdot\delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)})\right] \le 2\cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1)\cdot \varepsilon_{\mathsf{row}} \ .$$
- Using Lemma 11.6, as well as the fact that $f'|^{\updownarrow}_{\alpha} = \hat{f}|^{\updownarrow}_{\alpha}$, we deduce that
$$\delta^{(*,e)}_{L_0' \times Z_{L_0}(L_1)}(f') = \mathop{\mathbf{E}}_{\alpha \in L_0'}\left[\delta^{(e)}_{Z_{L_0}(L_1)}(f'|^{\updownarrow}_{\alpha})\right] = \mathop{\mathbf{E}}_{\alpha \in L_0'}\left[\delta^{(e)}_{Z_{L_0}(L_1)}(\hat{f}|^{\updownarrow}_{\alpha})\right] \le D(\lceil \tfrac{\kappa}{2} \rceil + \gamma)\cdot \varepsilon_{\mathsf{col}} \ .$$

Observing that

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

we can now invoke Theorem 9.2 with $m = |L_0'|$, $n = |Z_{L_0}(L_1)|$, $A = L_0'$, $B = Z_{L_0}(L_1)$, $g = f'$, and the d and e of this proof (indeed, because $\eta \ge 2$ from Equation 6, we get that $\frac{d}{m} + \frac{e}{n} \le \frac{1}{4} + \frac{1}{2} < 1$, so that the hypothesis of the theorem is satisfied), we get:

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

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 $\mathrm{RM}(\mathbb{F}_{2^\ell}, L'_0 \times Z_{L_0}(L_1), (d,e))$ closest to f'; in particular, $\delta^{(d,e)}_{L'_0 \times Z_{L_0}(L_1)}(f') = \Delta_{L'_0 \times Z_{L_0}(L_1)}(f', Q)$;
- for every $\beta \in L_1$, define the function $\hat{f}|^{\leftrightarrow\prime}_{Z_{L_0}(\beta)} \colon L'_0 \to \mathbb{F}_{2^\ell}$ by $\hat{f}|^{\leftrightarrow\prime}_{Z_{L_0}(\beta)} := (\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)})|_{L'_0}$ (and recall that L_β is the domain of $\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)}$, $L'_0 \subseteq L_\beta$, and $|L'_0| = |L_\beta|/2$); and
- for every $\beta \in L_1$, let $Q_{Z_{L_0}(\beta)}(x)$ be the polynomial over $\mathbb{F}_{2^\ell}$ corresponding to a codeword in $\mathrm{RS}(\mathbb{F}_{2^\ell}, L_\beta, d)$ closest to $\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)}$; in particular, $\delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)}) = \Delta_{L_\beta}(\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)}, Q_{Z_{L_0}(\beta)})$.

For every $\beta' \in Z_{L_0}(L_1)$, 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 $(\alpha,\beta') \in T$, $\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta')$:

$$\begin{aligned}
\Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta')\Big] &= \Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\Big|\, \beta' \text{ is bad}\Big] \cdot \Pr_{\beta'\in Z_{L_0}(L_1)}\Big[\beta' \text{ is bad}\Big] \\
&\quad + \Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\Big|\, \beta' \text{ is good}\Big] \cdot \Pr_{\beta'\in Z_{L_0}(L_1)}\Big[\beta' \text{ is good}\Big] \\
&\leq 1 \cdot \Pr_{\beta'\in Z_{L_0}(L_1)}\Big[\beta' \text{ is bad}\Big] \\
&\quad + \Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\Big|\, \beta' \text{ is good}\Big] \cdot 1 \\
&= \Pr_{\beta'\in Z_{L_0}(L_1)}\Big[\beta' \text{ is bad}\Big] + \Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\Big|\, \beta' \text{ is good}\Big] \ .
\end{aligned} \tag{15}$$

We begin with a bound on the first summand Equation 15. Observe that, for every $\beta' \in Z_{L_0}(L_1)$, 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,

$$\frac{2^{\eta-1}-1}{2^{\eta-1}} \leq \Delta_{L'_0}\big(Q_{\beta'}(x), Q(x,\beta')\big) \leq \Delta_{L'_0}\big(Q_{\beta'}(x), \hat{f}|^{\leftrightarrow\prime}_{Z_{L_0}(\beta)}\big) + \Delta_{L'_0}\big(\hat{f}|^{\leftrightarrow\prime}_{Z_{L_0}(\beta)}, Q(x,\beta')\big) \ .$$

For reasons that will become clear shortly, we try to understand the expectation, taken over a random $\beta' \in Z_{L_0}(L_1)$, of the above two distances.

From $L'_0 \subseteq L_\beta$ and $|L'_0| = |L_\beta|/2$, we obtain that

$$\Delta_{L'_0}\big(Q_{Z_{L_0}(\beta)}(x), \hat{f}|^{\leftrightarrow\prime}_{Z_{L_0}(\beta)}\big) \leq 2 \cdot \Delta_{L_\beta}\big(Q_{Z_{L_0}(\beta)}(x), \hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)}\big) = 2 \cdot \delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)}) \ ,$$

so that

$$\mathop{\mathbf{E}}_{\beta\in L_1}\Big[\Delta_{L'_0}\big(Q_{Z_{L_0}(\beta)}(x), \hat{f}|^{\leftrightarrow\prime}_{Z_{L_0}(\beta)}\big)\Big] \leq 2 \cdot \mathop{\mathbf{E}}_{\beta\in L_1}\Big[\delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)})\Big] \ .$$

Next, observe that, for every $\beta' \in Z_{L_0}(L_1)$, $\hat{f}|^{\leftrightarrow\prime}_{\beta'} = f'|^{\leftrightarrow}_{\beta'}$, because f' is simply the restriction of $\hat{f}$ to $L'_0 \times Z_{L_0}(L_1)$ and $\hat{f}|^{\leftrightarrow\prime}_{\beta}$ is the restriction of $\hat{f}|^{\leftrightarrow}_{\beta}$ to L'_0. Hence, for every $\beta' \in Z_{L_0}(L_1)$: it holds that $\Delta_{L'_0}\big(\hat{f}|^{\leftrightarrow\prime}_{\beta'}, Q(x,\beta')\big) = \Delta_{L'_0}\big(f'|^{\leftrightarrow}_{\beta'}, Q(x,\beta')\big)$, so that

$$\mathop{\mathbf{E}}_{\beta'\in Z_{L_0}(L_1)}\Big[\Delta_{L'_0}\big(\hat{f}|^{\leftrightarrow\prime}_{\beta'}, Q(x,\beta')\big)\Big] = \mathop{\mathbf{E}}_{\beta'\in Z_{L_0}(L_1)}\Big[\Delta_{L'_0}\big(f'|^{\leftrightarrow}_{\beta'}, Q(x,\beta')\big)\Big]$$

$$
\begin{aligned}
&= \frac{1}{|Z_{L_0}(L_1)|} \cdot \sum_{\beta' \in Z_{L_0}(L_1)} \Delta_{L_0'}\big(f'|^{\leftrightarrow}_{\beta'}, Q(x,\beta')\big) \\
&= \frac{1}{|Z_{L_0}(L_1)|} \cdot \sum_{\beta' \in Z_{L_0}(L_1)} \frac{\left|\left\{\alpha \in L_0' \,:\, Q(\alpha,\beta') \neq f'|^{\leftrightarrow}_{\beta'}(\alpha)\right\}\right|}{|L_0'|} \\
&= \frac{1}{|Z_{L_0}(L_1)|} \cdot \sum_{\beta' \in Z_{L_0}(L_1)} \frac{\left|\left\{\alpha \in L_0' \,:\, Q(\alpha,\beta') \neq f'(\alpha,\beta')\right\}\right|}{|L_0'|} \\
&= \frac{\left|\left\{(\alpha,\beta') \in L_0' \times Z_{L_0}(L_1) \,:\, Q(\alpha,\beta') \neq f'(\alpha,\beta')\right\}\right|}{|L_0'| \cdot |Z_{L_0}(L_1)|} \\
&= \Delta_{L_0' \times Z_{L_0}(L_1)}(f', Q) \\
&= \delta^{(d,e)}_{L_0' \times Z_{L_0}(L_1)}(f') \ .
\end{aligned}
$$

Hence, we get:

$$
\begin{aligned}
&\Pr_{\beta \in L_1}\Big[Z_{L_0}(\beta) \text{ is bad}\Big] \\
\leq\; &\Pr_{\beta \in L_1}\left[\Delta_{L_0'}\big(Q_{Z_{L_0}(\beta)}(x), \hat{f}|^{\leftrightarrow'}_{Z_{L_0}(\beta)}\big) + \Delta_{L_0'}\big(\hat{f}|^{\leftrightarrow'}_{Z_{L_0}(\beta)}, Q(x, Z_{L_0}(\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}\Big[\Delta_{L_0'}\big(Q_{Z_{L_0}(\beta)}(x), \hat{f}|^{\leftrightarrow'}_{Z_{L_0}(\beta)}\big) + \Delta_{L_0'}\big(\hat{f}|^{\leftrightarrow'}_{Z_{L_0}(\beta)}, Q(x, Z_{L_0}(\beta))\big)\Big] \quad \text{(via Markov's inequality)} \\
\leq\; &\frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \mathop{\mathbf{E}}_{\beta \in L_1}\Big[\Delta_{L_0'}\big(Q_{Z_{L_0}(\beta)}(x), \hat{f}|^{\leftrightarrow'}_{Z_{L_0}(\beta)}\big) + \Delta_{L_0'}\big(\hat{f}|^{\leftrightarrow'}_{Z_{L_0}(\beta)}, Q(x, Z_{L_0}(\beta))\big)\Big] \\
=\; &\frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \left(\mathop{\mathbf{E}}_{\beta \in L_1}\Big[\Delta_{L_0'}\big(Q_{Z_{L_0}(\beta)}(x), \hat{f}|^{\leftrightarrow'}_{Z_{L_0}(\beta)}\big)\Big] + \mathop{\mathbf{E}}_{\beta \in L_1}\Big[\Delta_{L_0'}\big(\hat{f}|^{\leftrightarrow'}_{Z_{L_0}(\beta)}, Q(x, Z_{L_0}(\beta))\big)\Big]\right) \\
\leq\; &\frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \left(2 \cdot \mathop{\mathbf{E}}_{\beta \in L_1}\Big[\delta^{(d)}_{L_\beta}\big(\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)}\big)\Big] + \delta^{(d,e)}_{L_0' \times Z_{L_0}(L_1)}(f')\right) \\
\leq\; &\frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \Big(2 \cdot D(\lfloor\tfrac{\kappa}{2}\rfloor - \gamma + \mu + 1) \cdot \varepsilon_{\mathsf{row}} + c_0 \cdot \Big(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}}\Big)\Big) \\
=\; &\frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \Big((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}}\Big) \\
=\; &\frac{2^{\eta-1}}{2^{\eta-1}-1} \cdot \Big((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\Big) \\
\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}{r} (2+2c_0) \cdot D(\lfloor\tfrac{\kappa}{2}\rfloor - \gamma + \mu + 1) \cdot \varepsilon_{\mathsf{row}} \\ \geq c_0 \cdot D(\lceil\tfrac{\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 15.

We now bound the second summand of Equation 15. 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:

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

$$
\begin{aligned}
&= \frac{\Pr_{(\alpha,\beta')\in T}\left[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \ \wedge\ \beta' \text{ is good}\right]}{\Pr_{(\alpha,\beta')\in T}\left[\beta' \text{ is good}\right]} \\
&= \frac{\left|\left\{(\alpha,\beta') \in T \,:\, \hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \ \wedge\ \beta' \text{ is good}\right\}\right|}{\left|\left\{(\alpha,\beta') \in T \,:\, \beta' \text{ is good}\right\}\right|} \\
&= \frac{\left|\left\{(\alpha,\beta') \in T \,:\, \hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \ \wedge\ \beta' \text{ is good}\right\}\right|}{\left|S \cup T\right|} \cdot \frac{\left|S \cup T\right|}{\left|\left\{(\alpha,\beta') \in T \,:\, \beta' \text{ is good}\right\}\right|} \\
&\leq \frac{\left|\left\{(\alpha,\beta') \in S \cup T \,:\, \hat{f}(\alpha,\beta') \neq Q(\alpha,\beta') \ \wedge\ \beta' \text{ is good}\right\}\right|}{\left|S \cup T\right|} \cdot \frac{\left|S \cup T\right|}{\left|\left\{(\alpha,\beta') \in T \,:\, \beta' \text{ is good}\right\}\right|} \\
&= \Pr_{(\alpha,\beta')\in S\cup T}\left[\hat{f}(\alpha,\beta') \neq Q(\alpha,\beta')\right] \cdot \frac{\left|S \cup T\right|}{|T_{\mathsf{good}}|} \\
&\leq \min\left\{\delta^{(d,*)}_{S\cup T}(\hat{f})\,,\, \delta^{(*,e)}_{S\cup T}(\hat{f})\right\} \cdot \frac{\left|S \cup T\right|}{|T_{\mathsf{good}}|} \\
&\leq \delta^{(d,*)}_{S\cup T}(\hat{f}) \cdot \frac{\left|S \cup T\right|}{|T_{\mathsf{good}}|} \\
&= \mathop{\mathbf{E}}_{\beta\in L_1}\left[\delta^{(d)}_{L_\beta}(\hat{f}|^{\leftrightarrow}_{Z_{L_0}(\beta)})\right] \cdot \frac{\left|S \cup T\right|}{|T_{\mathsf{good}}|} \\
&\leq D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1) \cdot \varepsilon_{\mathsf{row}} \cdot \frac{\left|S \cup T\right|}{|T_{\mathsf{good}}|} \\
&\leq 2 \cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1) \cdot \varepsilon \cdot \frac{\left|S \cup T\right|}{|T_{\mathsf{good}}|} \quad .
\end{aligned}
$$

For every $\beta \in L_1$:

- by the definition of T, the $Z_{L_0}(\beta)$-row of T is $L_0 + \beta + \mathcal{O}_L$, an affine shift of L_0 by $\beta + \mathcal{O}_L$; and
- by Lemma 11.8, the $Z_{L_0}(\beta)$-row of $S \cup T$ is L_β;

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

$$
\frac{|S \cup T|}{|T_{\mathsf{good}}|} = \frac{|S \cup T|}{|T|} \cdot \frac{|T|}{|T_{\mathsf{good}}|} = 2^{\mu+1} \cdot \frac{1}{\Pr_{\beta'\in Z_{L_0}(L_1)}\left[\beta' \text{ is good}\right]} = 2^{\mu+1} \cdot \frac{1}{1 - \Pr_{\beta'\in Z_{L_0}(L_1)}\left[\beta' \text{ is bad}\right]} \quad .
$$

Thus, the second summand of Equation 15 can be upper bounded as follows:

$$
\Pr_{(\alpha,\beta')\in T}\left[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta') \,\middle|\, \beta' \text{ is good}\right] \leq \frac{2^{\mu+2} \cdot D(\lfloor \frac{\kappa}{2} \rfloor - \gamma + \mu + 1)}{1 - \Pr_{\beta'\in Z_{L_0}(L_1)}\left[\beta' \text{ is bad}\right]} \cdot \varepsilon \quad .
$$

In conclusion, we have established that:

$$
\Pr_{(\alpha,\beta')\in T}\left[\hat{p}(\alpha,\beta') \neq Q(\alpha,\beta')\right] \leq \Pr_{\beta'\in Z_{L_0}(L_1)}\left[\beta' \text{ is bad}\right] + \frac{2^{\mu+2} \cdot D(\lfloor \frac{\kappa}{2} \rfloor - \gamma + \mu + 1)}{1 - \Pr_{\beta'\in Z_{L_0}(L_1)}\left[\beta' \text{ is bad}\right]} \cdot \varepsilon \ , \qquad (16)
$$

where

$$\Pr_{\beta'\in Z_{L_0}(L_1)}\big[\beta' \text{ is bad}\big] \leq \frac{2^\eta}{2^{\eta-1}-1}\cdot\max\{(2+2c_0)\cdot D(\lfloor\tfrac{\kappa}{2}\rfloor-\gamma+\mu+1)\,,\, c_0\cdot D(\lceil\tfrac{\kappa}{2}\rceil+\gamma)\}\cdot\varepsilon\ . \quad (17)$$

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

$$\Pr_{(\alpha,\beta')\in T}\Big[\hat{p}(\alpha,\beta')\neq Q(\alpha,\beta')\Big]\leq c^{\log\kappa}\cdot\varepsilon\ . \quad (18)$$

We observe that, because $\kappa>\kappa_0$,

$$D(\lfloor\tfrac{\kappa}{2}\rfloor-\gamma+\mu+1)\leq c^{\log(\lfloor\frac{\kappa}{2}\rfloor-\gamma+\mu+1)}\leq\frac{c^{\log\kappa}}{c^{\log\left(\frac{\kappa_0+\frac{3+(-1)^{\kappa_0 \bmod 2}}{2}}{\lfloor(\kappa_0+\frac{3+(-1)^{\kappa_0 \bmod 2}}{2})/2\rfloor-\gamma+\mu+1}\right)}}=\frac{c^{\log\kappa}}{c^{e_1}} \quad (19)$$

and

$$D(\lceil\tfrac{\kappa}{2}\rceil+\gamma)\leq c^{\log(\lceil\frac{\kappa}{2}\rceil+\gamma)}\leq\frac{c^{\log\kappa}}{c^{\log\left(\frac{\kappa_0+\frac{1+(-1)^{\kappa_0 \bmod 2}}{2}}{\lceil(\kappa_0+\frac{1+(-1)^{\kappa_0 \bmod 2}}{2})/2\rceil+\gamma}\right)}}=\frac{c^{\log\kappa}}{c^{e_2}}\ . \quad (20)$$

We have also used the fact (from Equation 6) that $\mu\geq\gamma$, $\frac{\kappa_0+\frac{3+(-1)^{\kappa_0 \bmod 2}}{2}}{\lfloor(\kappa_0+\frac{3+(-1)^{\kappa_0 \bmod 2}}{2})/2\rfloor-\gamma+\mu+1}>1$, and $\frac{\kappa_0+\frac{1+(-1)^{\kappa_0 \bmod 2}}{2}}{\lceil(\kappa_0+\frac{1+(-1)^{\kappa_0 \bmod 2}}{2})/2\rceil+\gamma}>1$.

Next, combining Equation 16 and Equation 17, by substituting the upper bounds from Equation 19 and Equation 20, and recalling that by assumption $c^{\log\kappa}\cdot\varepsilon\leq 1$, we get:

$$\begin{aligned}\Pr_{(\alpha,\beta')\in T}&\Big[\hat{p}(\alpha,\beta')\neq Q(\alpha,\beta')\Big]\\ &\leq\Bigg(\frac{2^\eta}{2^{\eta-1}-1}\cdot\max\{(2+2c_0)\cdot c^{-e_1}\,,\,c_0\cdot c^{-e_2}\}\\ &\qquad+\frac{2^{\mu+2}\cdot c^{-e_1}}{1-\frac{2^\eta}{2^{\eta-1}-1}\cdot\max\{(2+2c_0)\cdot c^{-e_1}\,,\,c_0\cdot c^{-e_2}\}}\Bigg)\cdot c^{\log\kappa}\cdot\varepsilon\ ,\end{aligned} \quad (21)$$

So, noting that

$$\max\{(2+2c_0)\cdot c^{-e_1}\,,\,c_0\cdot c^{-e_2}\}\leq(2+2c_0)c^{\max\{-e_1\,,-e_2\}}=(2+2c_0)c^{-\min\{e_1\,,e_2\}}\ ,$$

it suffices to require that

$$\frac{a}{x(c)}+\frac{b\cdot\frac{1}{x(c)}}{1-\frac{a}{x(c)}}\leq 1\ ,\ \text{where}\ \begin{cases}a\stackrel{\text{def}}{=}\dfrac{2^{\eta+1}(1+c_0)}{2^{\eta-1}-1}>0\\ b\stackrel{\text{def}}{=}2^{\mu+2}>0\\ x(c)\stackrel{\text{def}}{=}c^{\min\{e_1\,,e_2\}}>a\end{cases}\ . \quad (22)$$

Solving the (second-degree) inequality in $x(c)$, we obtain that the solutions with $x(c)>a$ are:

$$x(c)\geq a+\frac{b}{2}+\frac{\sqrt{b(b+4a)}}{2}\ . \quad (23)$$

Therefore, we choose c so that $x(c) = a + \frac{b}{2} + \frac{\sqrt{b(b+4a)}}{2}$. In other words, we choose c such that

$$c \stackrel{\text{def}}{=} \left(a + \frac{b}{2} + \frac{\sqrt{b(b+4a)}}{2}\right)^{\frac{1}{\min\{e_1, e_2\}}} , \tag{24}$$

and this choice of c will ensure that Equation 18 is satisfied.

Step 3: From bivariate $\hat{p}$ to univariate p. Let $P(x)$ be the polynomial over $\mathbb{F}_{2^\ell}$ defined by $P(x) := Q(x, Z_{L_0}(x))$; its degree is at most $|L|/2^\eta - 1$, as is possible to see by direct computation:

$$\begin{aligned}
\deg P &= \deg_x(Q) + \deg(Z_{L_0}) \cdot \deg_y(Q) \\
&\leq d + |L_0| \cdot e \\
&= \left(\frac{|L_\beta|}{2^\eta} - 1\right) + |L_0| \cdot \left(\frac{|Z_{L_0}(L_1)|}{2^\eta} - 1\right) \\
&= \left(\frac{|L_\beta|}{2^\eta} - 1\right) + |L_0| \cdot \left(\frac{|L_1|}{2^\eta} - 1\right) \\
&= \left(2^{\dim(L_\beta)-\eta} - 1\right) + 2^{\dim(L_0)} \cdot \left(2^{\dim(L_1)-\eta} - 1\right) \\
&= \left(2^{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1 - \eta} - 1\right) + 2^{\lfloor \kappa/2 \rfloor - \gamma} \cdot \left(2^{\lceil \kappa/2 \rceil + \gamma - \eta} - 1\right) \\
&= (2^{\kappa-\eta} - 1) + (2^{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1 - \eta} - 2^{\lfloor \kappa/2 \rfloor - \gamma}) \\
&= (2^{\kappa-\eta} - 1) + 2^{\lfloor \kappa/2 \rfloor - \gamma} \cdot (2^{\mu+1-\eta} - 1) \qquad \text{(from Equation 6, } \mu + 1 \leq \eta) \\
&\leq (2^{\kappa-\eta} - 1) + 0 \\
&= \frac{|L|}{2^\eta} - 1 \ .
\end{aligned}$$

We have also used the fact, guaranteed by Equation 6, that $\lfloor (\kappa_0 + 1)/2 \rfloor - \gamma + \mu + 1 > \eta$ and $\lceil (\kappa_0 + 1)/2 \rceil + \gamma > \eta$. 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 18 and from the definition of T as the set $T = \{(\tau, Z_{L_0}(\tau)) : \tau \in L\}$, we deduce that for all but at most a $(c^{\log \kappa} \cdot \varepsilon)$-fraction of the elements in L it holds that

$$p(\tau) = \hat{p}(\tau, Z_{L_0}(\tau)) = Q(\tau, Z_{L_0}(\tau)) = P(\tau),$$

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 11.9 (Completeness)**.** In the above proof, we have not used the constraint $\mu + 1 \geq \eta$ from Equation 6. 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 $Z_{L_0}(\beta)$-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 $\mathrm{RS}(\mathbb{F}_{2^\ell}, Z_{L_0}(L_1), |Z_{L_0}(L_1)|/2^\eta - 1)$ for every $\beta \in L_1$ and $\alpha \in L_0'$.

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

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

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 $\beta' = Z_{L_0}(\alpha)$, so that

$$\hat{p}(\alpha, \beta') = \hat{p}(\alpha, Z_{L_0}(\alpha)) = p(\alpha) = P(\alpha) = Q(\alpha, Z_{L_0}(\alpha)) = Q(\alpha, \beta') \ .$$

Therefore:

- for each $\beta \in L_1$, the $Z_{L_0}(\beta)$-row of $S \cup T$ is L_β and, by construction, the restriction of $\hat{f}$ to L_β is the evaluation of $Q(x, Z_{L_0}(\beta))$ over L_β; hence, because $\mu + 1 \geq \eta$,

$$\deg(Q(x, Z_{L_0}(\beta))) \leq \deg_x(Q) < |L_0| = \frac{|L_\beta|}{2^{\mu+1}} \leq \frac{|L_\beta|}{2^\eta} \ .$$

- for each $\alpha \in L_0'$, the α-column of $\hat{f}$ is $Z_{L_0}(L_1)$ and, by construction, the restriction of $\hat{f}$ to $Z_{L_0}(L_1)$ is the evaluation of $Q(\alpha, y)$ on $Z_{L_0}(L_1)$; hence,

$$\deg(Q(\alpha, y)) \leq \deg_y(Q) < \frac{|Z_{L_0}(L_1)|}{2^\eta} \ .$$

This completes the proof of completeness.

11.1.4 Soundness for concrete dimensions

While the soundness analysis of Section 11.1.3 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 16 and Equation 17. From these two equations, we deduce that we wish $D(\kappa)$ to be the smallest function that satisfies the following recursive inequality:

$$\begin{aligned}
&\frac{2^\eta}{2^{\eta-1}-1} \cdot \max\{(2+2c_0) \cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1)\, , \, c_0 \cdot D(\lceil \tfrac{\kappa}{2} \rceil + \gamma)\} \\
&\qquad + \frac{2^{\mu+2} \cdot D(\lfloor \frac{\kappa}{2} \rfloor - \gamma + \mu + 1)}{1 - \frac{1}{D(\kappa)} \cdot \frac{2^\eta}{2^{\eta-1}-1} \cdot \max\{(2+2c_0) \cdot D(\lfloor \frac{\kappa}{2} \rfloor - \gamma + \mu + 1)\, , \, c_0 \cdot D(\lceil \frac{\kappa}{2} \rceil + \gamma)\}} \leq D(\kappa) \ .
\end{aligned}$$

Furthermore, we must have $D(\kappa) > \frac{2^\eta}{2^{\eta-1}-1} \cdot \max\{(2+2c_0) \cdot D(\lfloor \frac{\kappa}{2} \rfloor - \gamma + \mu + 1)\, , \, c_0 \cdot D(\lceil \frac{\kappa}{2} \rceil + \gamma)\}$. Solving the inequality and then picking the smallest valid solution (in a similar manner to the way we solved the inequality in Section 11.1.3), we obtain the following recursive definition for $D(\kappa)$:

$$D(\kappa) = \begin{cases} \text{if } \kappa \leq \kappa_0 : & 1 \\ \text{if } \kappa > \kappa_0 : & a(\kappa) + \dfrac{b(\kappa)}{2} + \dfrac{\sqrt{b(\kappa)(b(\kappa)+4a(\kappa))}}{2} \end{cases} , \tag{25}$$

where

$$\begin{aligned} a(\kappa) &:= \frac{2^\eta}{2^{\eta-1}-1} \cdot \max\{(2+2c_0)\cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1)\,,\, c_0 \cdot D(\lceil \tfrac{\kappa}{2} \rceil + \gamma)\} \\ b(\kappa) &:= 2^{\mu+2} \cdot D(\lfloor \tfrac{\kappa}{2} \rfloor - \gamma + \mu + 1) \end{aligned}$$

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

$$\log c(\kappa) \stackrel{\text{def}}{=} \frac{\log D(\kappa)}{\log \kappa} . \tag{26}$$

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).

11.2 Soundness Analysis of $V_{\text{RS},<}$

The (strong) PCPP verifier $V_{\text{RS},<}$ (see Algorithm 15) tests proximity to $\text{RS}(\mathbb{F}, S, d)$ for $d < |S|/2^\eta - 1$; it is constructed using the (strong) PCPP verifier $V_{\text{RS},=}$.

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

Lemma 11.10. *If $V_{\text{RS},=}$ has monotone soundness function $s_{\text{RS},=}(\delta, n)$, then $V_{\text{RS},<}$ has soundness function*

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

Proof. Fix an explicit input $(\mathbb{F}, S, d)$ and an implicit input $p \colon S \to \mathbb{F}$ to $V_{\text{RS},<}$ 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 $V_{\text{RS},<}$. Define $\delta := \Delta_S\big(p, \text{RS}(\mathbb{F}, S, d)\big)$ and assume that $\delta > 0$.

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

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

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

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

  Then, p is both δ-far from $\text{RS}(\mathbb{F}, S, d)$ and $\mu\delta$-close to $\text{RS}(\mathbb{F}, S, d_{\kappa,\eta})$. 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 $\left(\frac{2^\eta-1}{2^\eta} - \mu\delta\right)$-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 $s_{\mathrm{RS},=}\left(\frac{2^\eta-1}{2^\eta} - \mu\delta, n\right)$, which, by the monotonicity of $s_{\mathrm{RS},=}$, is at least $s_{\mathrm{RS},=}\left(\frac{2^\eta-1}{2^\eta}\delta - \mu\delta, n\right)$.

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 11.11. *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 11.4 and Lemma 11.10. □

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

The (strong) PCPP verifier $V_{\mathrm{RS},>}$ (see Algorithm 14) 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 $V_{\mathrm{RS},<}$.

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 11.12. *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*

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

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 $d_{\kappa,\eta} := |S|/2^\eta - 1$. 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 $\pi = \left((p_0, \pi_0), \ldots, (p_{2^\eta-1}, \pi_{2^\eta-1})\right)$ be any proximity proof given to $V_{\mathrm{RS},>}$. Define $\delta := \Delta_S\left(p, \mathrm{RS}(\mathbb{F}, S, d)\right)$ 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 $s_{\mathrm{RS},=}\left(\frac{\tau\delta}{2^\eta}, n\right)$; 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 $s_{\mathrm{RS},<}\left(\frac{\tau\delta}{2^\eta}, n\right)$.

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

Corollary 11.13. *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 11.10, we know that $s_{\mathrm{RS},<}(\delta, n) \leq s_{\mathrm{RS},=}(\delta, n)$, so, from Lemma 11.12, 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 11.11, 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 $s_{\mathrm{RS},>}(\delta, n)$. □

11.4 Soundness Analysis of V_{RS}

The (strong) PCPP verifier V_{RS} (see Algorithm 13) 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 $V_{\mathrm{RS},>}$.

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

Lemma 11.14. *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*

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

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 11.15. *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 11.10 and Lemma 11.12 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 11.14, we know that $s_{\mathrm{RS}}(\delta, n) \geq s_{\mathrm{RS},>}(\delta, n)$, which can be lower bounded by Corollary 11.13, yielding the claimed lower bound. (And thus we have established the first lower bound from Theorem 11.1.) □

11.5 Soundness Analysis of V_{VRS}

The (strong) PCPP verifier V_{VRS} (see Algorithm 12) 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}.

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

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

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 $Z_H : \mathbb{F} \to \mathbb{F}$ be the vanishing polynomial for the subspace H (see Algorithm 21), 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 11.17. *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 11.16, know that $s_{\mathrm{VRS}}(\delta, n) \geq \min\{s_{\mathrm{RS}}(\tau\delta, n)\, ,\, (1 - \tau)\delta\}$. Moreover, from Corollary 11.15, 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 11.1.) □

12 Proof of Theorem 4

In this section we prove Theorem 4, discussed in Section 1.6 and formally stated in Section 2.5.

Proof of Theorem 4. Because the reductions of [BSCGT13] preserve the properties we are going to prove (see Remark 12.1), it suffices to concentrate on our PCP construction for sACSP from the proof of Theorem 8.1.

We thus proceed as follows. We formally introduce universal arguments, along with our relaxation of *almost* universal arguments (Section 12.1). We then prove that the PCPs from Theorem 8.1 have the relatively-efficient oracle construction property (Section 12.2), have the non-adaptive verifier property (Section 12.3), have the efficient reverse-sampling property (Section 12.4), and the "explicit" proof of knowledge property (Section 12.5). Finally, we show how to construct almost universal arguments using PCPs with these four properties (Section 12.6). Throughout, when providing pseudocode, we freely use procedures defined in Section B.3.

Also, it suffices to prove that our PCP construction for sACSP can be used to construct almost universal arguments, because [Mic00, Val08, CT10, BCCT12a, CT12] only need a subset of the properties needed for almost universal arguments.[22] □

We note that if one wishes to use a PCP system without an efficient reverse sampler, not all is lost: as in prior analyses [Kil92, Mic00], one can still obtain succinct arguments of knowledge for NP by assuming the existence of *strong* collision-resistant hash functions.

Remark 12.1 (choice of computation model)**.** It suffices to prove the four required properties for our PCP system for (an arbitrary choice of parameters for) sACSP, because any reasonable Levin reduction to sACSP will preserve the four properties;[23] in particular, so will the Levin reductions from bounded halting problem on RAMs studied by Ben-Sasson et al. [BSCGT13]. Therefore, our focus will be on the PCP system $(P_{\text{sACSP}}, V_{\text{sACSP}})$ that we construct for sACSPs, 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 sACSP.

Remark 12.2 (examples of positive applications)**.** Succinct arguments for NP have been used to achieve cryptographic tasks potentially very useful in practice. Indeed, most such "positive" applications of succinct arguments only consider the verification of computations that lie in NP; in particular, the "full power" of universal arguments (or even almost universal arguments) is usually not needed.[24]

Theorem 4 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:

[22] More precisely, these latter works require a stronger notion of a non-adaptive verifier than what is required for universal arguments. Namely, (standard) non-adaptivity only requires that the verifier algorithm can be split into a query algorithm and a decision algorithm. A stronger notion of adaptivity further requires that the query algorithm only needs as input a bound on the computation (rather than the actual statement to be proved). Our PCP construction for sACSP does enjoy this stronger notion of "statement-oblivious" non-adaptivity.

[23] Even *computational* Levin reductions, when properly defined, will preserve these properties; see [BSCGT13].

[24] An exception is the work of Chase and Visconti [CV12], who have shown how to "boost" the knowledge property of a universal argument in order to obtain a secure implementation of a database commitment functionality that hides the input size.

- **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.[25]

- **Proof-carrying data.** Chiesa and Tromer [CT10, CT12] have shown how to construct non-interactive succinct arguments of knowledge that can be "recursively composed",[26] 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 12.3 (computational overheads)**.** The construction of universal arguments starting from a PCP system (with certain properties) and a collision-resistant hash-function family [BG08] 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 [BG08], 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.[27]

Other constructions of succinct arguments from PCPs are not as "light"; for example, in [DCL08, BCCT12a, DFH12, GLR11], one must also use a private information retrieval scheme.

12.1 Universal Arguments and Our Relaxation

A *succinct argument for* NP is a computationally-sound interactive proof for NP where, for a given NP language L, in order to check membership of a given instance y in L, the verifier only needs to run in time that is bounded by $p(\kappa, |y|)$ and the prover only needs to run in time that is

[25]Sometimes one is also interested in privacy. Such a property can be achieved by properly combining the succinct argument with a fully-homomorphic encryption scheme [Gen09]. Fully-homomorphic encryption is another computationally-heavy primitive, whose efficiency has been studied by, e.g., [GH11b, GH11a, BGV12, GSS12b, GSS12a]. However, quasilinear-time reductions that are compatible with fully-homomorphic encryption are not known; see [BSCGT13].

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

[27]It is of course possible to extend the discussion of Section 2.2 about concrete-efficiency thresholds to the case of succinct arguments.

bounded by $p(\kappa, T)$, 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. A succinct argument *of knowledge* for NP is a succinct argument for NP where soundness is strengthened to a proof-of-knowledge property.

Roughly, a *universal argument* [BG08] is a succinct argument of knowledge for NEXP, where, in order to check membership of an instance y in $L \in \mathsf{NEXP}$, the verifier only needs to run in time that is bounded by $p(\kappa, |y|, \log T)$ and the prover only needs to run in time that is bounded by $p(\kappa, T)$, where T is the time to (non-deterministically) evaluate the verification relation for L on input y, p is a fixed polynomial independent of L, and κ is a security parameter.

The difference between universal arguments and *almost* universal arguments, achieved by our PCPs, relates to the efficiency of the knowledge extractor. Details follow.

Barak and Goldreich [BG08] consider a language defined as follows (once adapted to the case of random-access machines):

Definition 12.4 (Universal Set). *The* **universal set***, denoted $S_{\mathcal{U}}$, is the set of all triples $y = (M, \mathtt{x}, T)$ such that M is (the description of) a two-tape random-access machine, $\mathtt{x}$ is a binary string, and T is a binary integer such that there is a binary string $\mathtt{w}$ for which M accepts $(\mathtt{x}, \mathtt{w})$ within T steps (where $\mathtt{x}$ is written on the input tape and $\mathtt{w}$ 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.*

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 $\mathtt{x}$ to the triple $(M_L, \mathtt{x}, 2^{|\mathtt{x}|})$, 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 12.5. *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 $y = (M, \mathtt{x}, T)$, the total time spent by the (probabilistic) verifier strategy V_{UA}, on common input y, is at most $p(|y|) = p(|M| + |\mathtt{x}| + \log T)$. 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 $\big((M, \mathtt{x}, T), \mathtt{w}\big) \in R_{\mathcal{U}}$,*
$$\Pr\Big[\langle P_{\mathsf{UA}}(\mathtt{w}), V_{\mathsf{UA}}\rangle(M, \mathtt{x}, T) = 1\Big] = 1 \ .$$
Furthermore, there exists a polynomial p such that for every $((M, \mathtt{x}, T), \mathtt{w}) \in R_{\mathcal{U}}$ the total time spent by $P_{\mathsf{UA}}(\mathtt{w})$, on common input $(M, \mathtt{x}, T)$, is at most
$$p\Big(|M| + |\mathtt{x}| + \mathsf{time}_M(\mathtt{x}, \mathtt{w})\Big) \leq p\big(|M| + |\mathtt{x}| + T\big) \ .$$

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 $(M, \mathtt{x}, T) \in \{0,1\}^\kappa - S_{\mathcal{U}}$,*
$$\Pr\Big[\langle \widetilde{P}_\kappa, V_{\mathsf{UA}}\rangle(M, \mathtt{x}, T) = 1\Big] < \frac{1}{\kappa^c} \ .$$

4. Weak proof of knowledge: *Implies computational soundness, and is stated below in Definition 12.6.*

Definition 12.6 (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 $y = (M, \mathtt{x}, T) \in \{0,1\}^\kappa$ 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 $\mathtt{w}$ for y.

In symbols:

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

if

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

then

$$\Pr_r\Big[\text{there exists } \mathtt{w} = \mathtt{w}_1 \cdots \mathtt{w}_T \in R_{\mathcal{U}}(y) \text{ such that, } \forall\, i \in [T]\,,\ E_{\mathsf{UA}}^{\langle \widetilde{P}_{\mathsf{UA},\kappa}\rangle}(y, i\,; r) = \mathtt{w}_i\Big] > \frac{1}{q_{\mathsf{UA}}(\kappa)} \quad .$$

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 $y = (M, \mathtt{x}, T)$ 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 12.7 ([BG08])**.** *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 12.8 (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 $y = (M, \mathtt{x}, T) \in \{0,1\}^\kappa$ 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 $\mathtt{w}$ for y, *In symbols:*

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

if

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

then

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

We thus define almost universal arguments as universal arguments where the proof of knowledge property is relaxed to that of Definition 12.8.

Definition 12.9. *An* **almost universal argument** *system* $(P_{\mathsf{UA}}, V_{\mathsf{UA}})$ *satisfies [BG08, 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 12.1 in Section 2, we will not consider a "natural" universal language (such as the universal language relative to random-access machines or relative to Turing machines), but we will concentrate on sACSPs because the properties we shall prove will remain "invariant" under appropriate Levin reductions. In such a case, instances consist of triples $y = (\mathsf{par}_{\mathrm{sACSP}}, \mathtt{x}, 1^t)$, where $\mathsf{par}_{\mathrm{sACSP}}$ is a choice of parameters for sACSP and $(\mathtt{x}, 1^t) \in$ sACSP$(\mathsf{par}_{\mathrm{sACSP}})$. Throughout, we fix a choice of $\mathsf{par}_{\mathrm{sACSP}}$, so that it will suffice to specify the pair $(\mathtt{x}, 1^t)$.

12.2 Relatively-Efficient Oracle Construction

The first property considered by Barak and Goldreich [BG08, 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 12.10 (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 $(\mathtt{x}, \mathtt{w}) \in R$, algorithm P outputs an oracle $\pi_{\mathtt{x}}$ that makes V always accept.*

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

Claim 12.11. *The PCP verifier V_{sACSP} has the relatively-efficient oracle construction property for the relation induced by the language* sACSP.

Proof. From Theorem 8.1 we know that the PCP prover P_{sACSP} 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!).

12.3 Non-Adaptive Verifier

The second property considered by Barak and Goldreich [BG08, Definiton 3.2, second item] for a PCP verifier is the non-adaptivity of the queries:

Definition 12.12 (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 the decision of $D(x,r,b_1\cdots b_{p(|x|)})$, where p is some fixed polynomial.*

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

Claim 12.13. *The PCP verifier V_{sACSP} is a non-adaptive PCP verifier.*

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

Definition 12.14 (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". Please refer to Figure 1 for an algorithmic reference of the construction of V_{sACSP}. (More specific references will be given in each of the proofs.) Also, throughout, we fix a specific choice of parameters $(\eta,\kappa_0,\gamma,\mu)$, which parametrize V_{sACSP}. Finally, we will not carry out a complexity analysis for the "non-adaptively decomposed" verifiers, as a detailed analysis for the "non-decomposed" verifiers was already carried out in Section C.

Remark 12.15. 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 12.41.)

12.3.1 Non-Adaptivity of $V_{\mathrm{RS},=}$

Lemma 12.16. *$V_{\mathrm{RS},=}$ is a non-adaptive (strong) PCPP verifier.*

Proof. The explicit input of $V_{\mathrm{RS},=}$ is $(I_\ell,\mathcal{B}_L,\mathcal{O}_L)$, 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 $\mathcal{B}_L=(a_1,\ldots,a_\kappa)$ and $\mathcal{O}_L$ are the basis and the offset of a κ-dimensional affine subspace $L\subseteq\mathbb{F}_{2^\ell}$; the implicit input of $V_{\mathrm{RS},=}$ is a function $p\colon L \to \mathbb{F}_{2^\ell}$. (For more details on the algorithm $V_{\mathrm{RS},=}$, see Algorithm 16.) The proof of proximity π of $V_{\mathrm{RS},=}$ 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) affine subspaces ; see Algorithm 7 (and references therein) for the formal definition and for the algorithm that computes the proof of proximity.[28] Finally, see Lemma C.1 for $\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)$, $\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)$, and $\mathsf{length}_{\mathrm{RS},=}(\ell,\kappa)$.

The algorithm $V_{\mathrm{RS},=}$ 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 $\mathrm{RS}(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1)$ 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 $Q_{\mathrm{RS},=}$ will be the algorithm that produces all the queries of the base case, while the decision algorithm $D_{\mathrm{RS},=}$ 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 $Q_{\mathrm{RS},=}$, the recursive nature of $V_{\mathrm{RS},=}$ 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 $Q_{\mathrm{RS},=}$. (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 $D_{\mathrm{RS},=}$, the recursive nature of $V_{\mathrm{RS},=}$ makes things slightly more complicated, because the interpolation performed by $D_{\mathrm{RS},=}$ (as part of its verification test) needs to know which affine subspace 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 $Q_{\mathrm{RS},=}$, and then for the algorithm $D_{\mathrm{RS},=}$.

The "hard work" (mainly, bookkeeping) in $Q_{\mathrm{RS},=}$ is carried out by an iterative procedure that we call Translate; on input an irreducible polynomial I_ℓ of degree ℓ, a basis $\mathcal{B}_L$ and offset $\mathcal{O}_L$ for a κ-dimensional affine subspace $L \subseteq \mathbb{F}_{2^\ell}$, 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 $(I_\ell, \mathcal{B}_L, \mathcal{O}_L)$ and random string r; the other two components, $\mathtt{text} \in \{$"`implicit-input`", "`proximity-proof`"$\}$ 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:

[28] In fact, for the purpose of this proof, we change slightly the definition of the proof of proximity. Specifically, we let f be a bivariate function over $S \cup T$, as opposed to only over S; the values of f on T will never be used, but including them facilitates bookkeeping of indices. (Bhattacharyya [Bha05] already employed this slight change in his implementation of $V_{\mathrm{RS},=}$, as can be verified by inspection of his code, and as reflected in the proof length analysis in [Bha05, Section 2.2.2], where the function f is reported to require space $|L_1| \cdot |L_\beta| = |L_1| \cdot (8 \cdot |L_0|) = 2^{\lceil \kappa/2 \rceil} \cdot (8 \cdot 2^{\lfloor k/2 \rfloor})$, instead of $|L_1| \cdot |L_\beta - (L_0 + \beta + \mathcal{O}_L)| = |L_1| \cdot (7 \cdot |L_0|)$.)

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

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

The algorithm for $\textsc{Translate}$ 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:

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

1. If $\kappa \le \kappa_0$, then do the following:
   (a) $\alpha_i := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_L, \mathcal{O}_L, i)$;
   (b) $\texttt{text} :=$ "`implicit-input`";
   (c) $\mathsf{elt} := \alpha_i$;
   (d) output $(\texttt{text}, \mathsf{elt}, i)$.
2. If $\kappa > \kappa_0$, then do the following:
   (a) $\mathcal{B}_{L_0} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma})$, $\mathcal{O}_{L_0} := \mathcal{O}_L$;
   (b) $\mathcal{B}_{L_0'} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu})$, $\mathcal{O}_{L_0'} := \mathcal{O}_L$;
   (c) $[Z_{L_0}]^{\mathsf{A}} := \textsc{FindSubspPoly}(I_\ell, \mathcal{B}_{L_0}, \mathcal{O}_{L_0})$;
   (d) if $r_1 = 0$, then do <u>row test</u> translation:
      i. $m := 1 + \lceil \kappa/2 \rceil + \gamma$;
      ii. $\mathcal{B}_{L_1} := (a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_\kappa)$, $\mathcal{O}_{L_1} := \mathcal{O}_L$;
      iii. $\beta := \textsc{GetRandElt}(I_\ell, \mathcal{B}_{L_1}, \mathcal{O}_{L_1}; r_2 \cdots r_{1+\lceil \kappa/2 \rceil + \gamma})$;
      iv. $\beta' := [Z_{L_0}]^{\mathsf{A}}(\beta)$;
      v. $\iota_\beta := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_{L_1}, \mathcal{O}_{L_1}, \beta)$;
      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 $\mathcal{B}_{L_\beta} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}, \beta + \mathcal{O}_{L_0'})$;
      viii. $\mathcal{O}_{L_\beta} := \mathcal{O}_{L_0'}$;
      ix. $(\texttt{text}', \mathsf{elt}', \iota) := \textsc{Translate}(I_\ell, \mathcal{B}_{L_\beta}, \mathcal{O}_{L_\beta}, r_{m+1} \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i)$;
      x. if $\texttt{text}' =$ "`implicit-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:
            – $\texttt{text} :=$ "`implicit-input`";
            – $\mathsf{elt} := \alpha$;
            – $i := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_L, \mathcal{O}_L, \alpha)$;
            – output $(\texttt{text}, \mathsf{elt}, i)$;
         D. if $\alpha' \neq \beta'$, then:
            – $\texttt{text} :=$ "`proximity-proof`";
            – $\mathsf{elt} := (\alpha, \beta')$;
            – $i := |L| + |L_\beta| \cdot (\iota_\beta - 1) + \iota$;
            – output $(\texttt{text}, \mathsf{elt}, i)$;
      xi. if $\texttt{text}' =$ "`proximity-proof`", then:
         A. $\texttt{text} := \texttt{text}'$;
         B. $\mathsf{elt} := \mathsf{elt}'$;
         C. $i = (\iota - |L_\beta| + |L|) + |L_\beta| \cdot |L_1| + (\iota_\beta - 1) \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \dim L_\beta)$;
         D. output $(\texttt{text}, \alpha, i)$;
   (e) if $r_1 = 1$, then do <u>column test</u> translation:
      i. $m := 1 + \lfloor \kappa/2 \rfloor - \gamma + \mu$;
      ii. $\alpha := \textsc{GetRandElt}(I_\ell, \mathcal{B}_{L_0'}, \mathcal{O}_{L_0'}; r_2 \cdots r_{1+\lfloor \kappa/2 \rfloor - \gamma + \mu})$;
      iii. $\alpha' := [Z_{L_0}]^{\mathsf{A}}(\alpha)$;
      iv. $\iota_\alpha := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_{L_0'}, \mathcal{O}_{L_0'}, \alpha)$;
      v. $\mathcal{B}_{Z_{L_0}(L_1)} := ([Z_{L_0}]^{\mathsf{A}}(a_{\lfloor \kappa/2 \rfloor - \gamma + 1}) + [Z_{L_0}]^{\mathsf{A}}(0_{\mathbb{F}_{2^\ell}}), \ldots, [Z_{L_0}]^{\mathsf{A}}(a_\kappa) + [Z_{L_0}]^{\mathsf{A}}(0_{\mathbb{F}_{2^\ell}}))$;

vi. $\mathcal{O}_{Z_{L_0}(L_1)} := [Z_{L_0}]^{\mathsf{A}}(\mathcal{O}_{L_1})$;

vii. $(\texttt{text}', \mathsf{elt}', \iota) :=$ TRANSLATE$(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, \mathcal{O}_{Z_{L_0}(L_1)}, r_{m+1} \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i)$;

viii. if $\texttt{text}' =$ "`implicit-input`", then:

A. parse elt' as an element $\beta' \in Z_{L_0}(L_1)$;

B. if $\alpha' = \beta'$, then:

– $\texttt{text} :=$ "`implicit-input`";

– $\mathsf{elt} := \alpha$;

– $i :=$ GETINDEXOFELT$(I_\ell, \mathcal{B}_L, \alpha)$;

– output $(\texttt{text}, \mathsf{elt}, i)$;

C. if $\alpha' \neq \beta'$, then:

– $\texttt{text} :=$ "`proximity-proof`";

– $\mathsf{elt} := (\alpha, \beta')$;

– $i := |L| + |L_\beta| \cdot (\iota - 1) + \iota_\alpha$;

– output $(\texttt{text}, \mathsf{elt}, i)$;

ix. if $\texttt{text}' =$ "`proximity-proof`", then:

A. $\texttt{text} := \texttt{text}'$;

B. $\mathsf{elt} := \mathsf{elt}'$;

C. $i = (\iota - |Z_{L_0}(L_1)| + |L|) + |L_\beta| \cdot |L_1| + |L_1| \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \dim L_\beta) + (\iota_\alpha - 1) \cdot \mathsf{length}_{\mathrm{RS},=}(\ell, \dim Z_{L_0}(L_1))$;

D. output $(\texttt{text}, \alpha, i)$.

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.

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

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

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

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

Lemma 12.17. *$V_{\mathrm{RS},<}$ is a non-adaptive (strong) PCPP verifier.*

Proof. The explicit input of $V_{\mathrm{RS},<}$ is $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d)$, where I_ℓ is an irreducible polynomial over $\mathbb{F}_2$ of degree ℓ with root x, which induces the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$, $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_S$ are a basis and offset of a κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$, 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 $p \colon S \to \mathbb{F}_{2^\ell}$. (For more details on the algorithm $V_{\mathrm{RS},<}$, see Algorithm 15.) 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 $\mathrm{RS}(\mathbb{F}_{2^\ell}, S, d_{\kappa,\eta})$; again, see Algorithm 6 for more details and for the algorithm that computes the proof of proximity. Finally, see Lemma C.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 $Q_{\mathrm{RS},=}$.[29]

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

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

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) $j := Q_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S, \mathcal{O}_S), r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i)$;
   (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) $j_0 := Q_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S, \mathcal{O}_S), r''_1 \cdots r''_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \iota)$;
   (c) if $0 < j_0 \leq |S|$, then $j := j_0$;
   (d) if $|S| < j_0 \leq |S| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa)$, 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.

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

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

[29] 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.)

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. $b_1 := D_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S, \mathcal{O}_S), r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha'_1 \cdots \alpha'_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)})$,
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) $j_0 := Q_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S, \mathcal{O}_S), r''_1 \cdots r''_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i)$;
   (b) if $0 < j_0 \leq |S|$, then:
      i. $\gamma_{j_0} := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, j_0)$;
      ii. reset α''_i to the new value $\gamma_{j_0}^{d_{\kappa,\eta}-d} \alpha''_i$.
7. $b_2 := D_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S, \mathcal{O}_S), r''_1 \cdots r''_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha''_1 \cdots \alpha''_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)})$.
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 12.16. □

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

Lemma 12.18. *$V_{\mathrm{RS},>}$ is a non-adaptive (strong) PCPP verifier.*

Proof. The explicit input of $V_{\mathrm{RS},>}$ is $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d)$, where I_ℓ is an irreducible polynomial over $\mathbb{F}_2$ of degree ℓ with root x, which induces the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$, $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_S$ are a basis and offset of a κ-dimensional affine subspace 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 $p \colon S \to \mathbb{F}_{2^\ell}$. (For more details on the algorithm $V_{\mathrm{RS},>}$, see Algorithm 14.) 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 $\deg P_i = d_i \leq d_{\kappa,\eta}$; see Algorithm 5 (and the references therein) for more details and for the algorithm that computes the proof of proximity. Finally, see Lemma C.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:

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

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. $j_0 := Q_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S, \mathcal{O}_S), r_1^{(z)} \cdots r^{(z)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i')$;
      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. $j_0 := Q_{\mathrm{RS},<}((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d_{\kappa,\eta,d}), r_1^{(m_{\kappa,\eta,d}+1)} \cdots r^{(m_{\kappa,\eta,d}+1)}_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})}, i')$;
      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. $\gamma := \textsc{GetRandElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S; r_1^{(\gamma)} \cdots r_\kappa^{(\gamma)})$;
      iii. $\iota_\gamma := \textsc{GetIndexOfElt}(I_t, \mathcal{B}_S, \mathcal{O}_S, \gamma)$;
      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. $\gamma := \textsc{GetRandElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S; r_1^{(\gamma)} \cdots r_\kappa^{(\gamma)})$;
      iii. $\iota_\gamma := \textsc{GetIndexOfElt}(I_t, \mathcal{B}_S, \mathcal{O}_S, \gamma)$;
      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:

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

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) $\gamma := \textsc{GetRandElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S; r_1^{(\gamma)} \cdots r_\kappa^{(\gamma)})$;
(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. $b_i := D_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S, \mathcal{O}_S), r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha_1^{(i)} \cdots \alpha^{(i)}_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)})$;
   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. $b_{m_{\kappa,\eta,d}} := D_{\mathrm{RS},<}((I_\ell, \mathcal{B}_S, \mathcal{O}_S), r_1^{(m_{\kappa,\eta,d})} \cdots r^{(m_{\kappa,\eta,d})}_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})}, \alpha_1^{(m_{\kappa,\eta,d})} \cdots \alpha^{(m_{\kappa,\eta,d})}_{\mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})})$;
   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. $b_i := D_{\mathrm{RS},=}((I_\ell, \mathcal{B}_S, \mathcal{O}_S), r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, \alpha_1^{(i)} \cdots \alpha^{(i)}_{\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa)})$;
   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 12.16 and Lemma 12.17. □

12.3.4 Non-Adaptivity of V_{RS}

Lemma 12.19. V_{RS} *is a non-adaptive (strong) PCPP verifier.*

Proof. The explicit input of V_{RS} is $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d)$, where I_ℓ is an irreducible polynomial over $\mathbb{F}_2$ of degree ℓ with root x, which induces the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$, $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_S$ are a basis and offset of a κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$, and d is a positive integer; the implicit input of V_{RS} is a function $p \colon S \to \mathbb{F}_{2^\ell}$. (For more details on the algorithm V_{RS}, see Algorithm 13.) The proof of proximity π of V_{RS} is parsed according to whether d is less than, equal to, or greater than $d_{\kappa,\eta} := |S|/2^\eta - 1 = 2^{\kappa-\eta} - 1$; see Algorithm 4 for the algorithm that computes the proof of proximity according to these three cases. Finally, see Lemma C.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:

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

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 $j := Q_{\mathrm{RS},<}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d)}, i\big)$.
   (c) If $d = d_{\kappa,\eta}$, then output $j := Q_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}, i\big)$.
   (d) If $d > d_{\kappa,\eta}$, then output $j := Q_{\mathrm{RS},>}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d), r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},>}(\ell,\kappa,d)}, i\big)$.

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

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

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

The correctness of both Q_{RS} and D_{RS} easily follows by inspection of the algorithms and the above discussion, as well as Lemma 12.16, Lemma 12.17, and Lemma 12.18. □

12.3.5 Non-Adaptivity of V_{VRS}

Lemma 12.20. *V_{VRS} is a non-adaptive (strong) PCPP verifier.*

Proof. The explicit input of V_{VRS} is $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \mathcal{B}_H, \mathcal{O}_H, d)$, where I_ℓ is an irreducible polynomial over $\mathbb{F}_2$ of degree ℓ with root x, which induces the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$, $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_S$ are a basis and offset of a κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$, $\mathcal{B}_H = (b_1, \ldots, b_\lambda)$ and $\mathcal{O}_H$ are a basis and offset of a λ-dimensional affine subspace $H \subseteq \mathbb{F}_{2^\ell}$, and d is a positive integer; the implicit input of V_{VRS} is a function $p\colon S \to \mathbb{F}_{2^\ell}$. (For more details on the algorithm V_{VRS}, see Algorithm 12.) 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 $\mathrm{RS}(\mathbb{F}_{2^\ell}, S, d - |H|)$; see Algorithm 3 for the algorithm that computes the proof of proximity. Finally, see Lemma C.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 $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d - |H|)$ (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:

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

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

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

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

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. $b_{\mathrm{RS}} := D_{\mathrm{RS}}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d - |H|), r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d-|H|)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d-|H|)}\big)$.
4. $r_1^{(\alpha)} \cdots r_\kappa^{(\alpha)} := r_1 \cdots r_\kappa$.
5. $\alpha := \textsc{GetRandElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S; r_1^{(\alpha)} \cdots r_\kappa^{(\alpha)})$.
6. $[Z_H]^{\mathsf{A}} := \textsc{FindSubspPoly}(I_\ell, \mathcal{B}_H, \mathcal{O}_H)$;
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 12.19. □

12.3.6 Non-Adaptivity of V_{aRS}

Lemma 12.21. *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 12.19. For more details on the algorithm V_{aRS}, see Algorithm 11. (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}; more details on the prover for V_{RS} can be found in Algorithm 4.) Finally, see Lemma C.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:

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

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

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

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

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 $\alpha_1^{(1)}\cdots\alpha_{\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)}^{(1)}\cdots\alpha_1^{(m)}\cdots\alpha_{\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)}^{(m)}$.
3. For $i=1,\dots,m$, do the following:
   (a) $r_1^{(i)}\cdots r_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)}^{(i)} := r_{(i-1)\cdot\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)+1}\cdots r_{i\cdot\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)}$;
   (b) $b_i := D_{\mathrm{RS}}\big((I_\ell,\mathcal{B}_S,\mathcal{O}_S,d),r_1^{(i)}\cdots r_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d)}^{(i)},\alpha_1^{(i)}\cdots\alpha_{\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d)}^{(i)}\big)$.
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 12.19. □

Note that, in our explicit description of the algorithm V_{RS} (from Algorithm 11), 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 11.2), 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.

12.3.7 Non-Adaptivity of V_{aVRS}

Lemma 12.22. *V_{aVRS} is a non-adaptive PCPP verifier.*

Proof. This proof is completely analogous to that of Lemma 12.21, 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 $s_{\mathrm{VRS}}(\delta, n)$ 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 12.20. For more details on the algorithm V_{aVRS}, see Algorithm 10. (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}; more details on the prover for V_{VRS} can be found in Algorithm 3.) Finally, see Lemma C.7 for $\mathsf{query}_{\mathrm{aVRS}}(\ell, \kappa, \lambda, d, \delta, s')$ and $\mathsf{rand}_{\mathrm{aVRS}}(\ell, \kappa, \lambda, d, \delta, s')$.

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:

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

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. $i_{\mathsf{new}} := i - z \cdot \mathsf{query}_{\mathrm{VRS}}(\ell, \kappa, \lambda, d)$.
4. $r_1^{(i_{\mathsf{new}})} \cdots r_{\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}^{(i_{\mathsf{new}})} := r_{(i_{\mathsf{new}}-1)\cdot\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)+1} \cdots r_{i_{\mathsf{new}}\cdot\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}$.
5. $j := Q_{\mathrm{VRS}}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, \mathcal{B}_H, \mathcal{O}_H, d), r_1^{(i_{\mathsf{new}})} \cdots r_{\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}^{(i_{\mathsf{new}})}, i_{\mathsf{new}}\big)$.
6. Output j.

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

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

1. $m := \left\lceil \frac{\log(1-s')}{\log(1-s_{\mathrm{VRS}}(\delta,2^\kappa))} \right\rceil$.
2. Parse $\alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{aVRS}}(\ell,\kappa,\lambda,d,\delta,s')}$ as $\alpha_1^{(1)} \cdots \alpha_{\mathsf{query}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}^{(1)} \cdots \alpha_1^{(m)} \cdots \alpha_{\mathsf{query}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}^{(m)}$.
3. For $i = 1, \ldots, m$, do the following:
   (a) $r_1^{(i)} \cdots r_{\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}^{(i)} := r_{(i-1)\cdot\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)+1} \cdots r_{i\cdot\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}$;
   (b) $b_i := D_{\mathrm{VRS}}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, \mathcal{B}_H, \mathcal{O}_H, d), r_1^{(i)} \cdots r_{\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}^{(i)}, \alpha_1^{(i)} \cdots \alpha_{\mathsf{query}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}^{(i)}\big)$.
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 12.20. □

12.3.8 Non-Adaptivity of V_{sACSP}

Lemma 12.23. V_{sACSP} *is a non-adaptive PCP verifier.*

Proof. The PCP verifier V_{sACSP} is given as input an instance $(\mathtt{x}, 1^t)$, allegedly in sACSP, and oracle access to a PCP proof π. (For more details on the algorithm V_{sACSP}, see Algorithm 9.) 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 $\mathrm{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$, π_1 is a proof of proximity for p_1 to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d)$, and π_{c} is a proof of proximity for $p_0 - p_{\mathtt{x}}$ to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, I_{t,\log|\mathtt{x}|}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$ for a certain degree d, function $p_{\mathtt{x}}$, and affine subspace $I_{t,\log|\mathtt{x}|}$; see Algorithm 2 for the algorithm that computes the PCP proof. Finally, see Lemma C.8 for $\mathsf{query}_{\mathrm{ACSP}}(\mathtt{x}, t)$, $\mathsf{rand}_{\mathrm{ACSP}}(\mathtt{x}, t)$, and $\mathsf{length}_{\mathrm{ACSP}}(\mathtt{x}, t)$.

The algorithm V_{sACSP} consists of five main steps: in the first step, V_{sACSP} generates the necessary "objects" needed for running subsequent steps; in the second step, V_{sACSP} tests proximity of p_0 to RS (using a specific proximity parameter δ_{RS} and target soundness $s'_{\mathrm{RS}} = 1/2$); in the third step, V_{sACSP} tests proximity of p_1 to VRS (using another specific proximity parameter δ_{VRS} and target soundness $s'_{\mathrm{VRS}} = 1/2$); in the fourth step, V_{sACSP} runs a consistency test between p_0 and p_1; in the fifth step, V_{sACSP} tests proximity of $p_0 - p_{\mathtt{x}}$ to VRS (using another specific proximity parameter δ_{c} and target soundness $s'_{\mathsf{c}} = 1/2$).

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

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

1. Parameter instantiation:
   (a) $I_t := \textsc{FindIrrPoly}(1^{f(t)})$;
   (b) $\mathcal{B}_{\mathbb{F}_t} := \textsc{FieldBasis}(I_t)$;
   (c) $(\mathcal{B}_{H_t}, \mathcal{O}_{H_t}) := \textsc{Find}\mathbf{H}(1^t)$;
   (d) for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $[N_{t,i}]^{\mathsf{A}} := \textsc{Find}\mathbf{N}(1^t, i)$;
   (e) $[P_t]^{\mathsf{A}} := \textsc{Find}\mathbf{P}(1^t)$;
   (f) $(\mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}) := \textsc{Find}\mathbf{I}(1^t, 1^{\log|\mathtt{x}|})$;
   (g) $d := \deg(P_t(x, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \ldots, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathsf{c}_{\mathbf{N}}(t)})}))$.
2. Deduce the amount of randomness and number of queries for the amplified verifiers:
   (a) $\delta_{\mathrm{RS}} := (8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$ 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) $\mathsf{nq}_{\mathrm{aVRS}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$;
   (f) $\mathsf{nr}_{\mathrm{aVRS}} := \mathsf{rand}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$;
   (g) $\delta_{\mathsf{c}} := (8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$ and $s'_{\mathsf{c}} := 0.5$;
   (h) $\mathsf{nq}_{\mathsf{c}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \log|\mathtt{x}|, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$;
   (i) $\mathsf{nr}_{\mathsf{c}} := \mathsf{rand}_{\mathrm{aVRS}}(f(t), f(t), \log|\mathtt{x}|, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$.
3. If $i \in \{1, \ldots, \mathsf{nq}_{\mathrm{aRS}}\}$, then:
   (a) $r'_1 \cdots r'_{\mathsf{nr}_{\mathrm{aRS}}} := r_1 \cdots r_{\mathsf{nr}_{\mathrm{aRS}}}$;
   (b) $j_{\mathsf{old}} := Q_{\mathrm{aRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}}), r'_1 \cdots r'_{\mathsf{nr}_{\mathrm{aRS}}}, i\big)$;
   (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) $j_{\mathsf{old}} := Q_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{H_t}, \mathcal{O}_{H_t}, d, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}}), r''_1 \cdots r''_{\mathsf{nr}_{\mathrm{aVRS}}}, i_{\mathsf{new}}\big)$;
   (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) $\alpha := \textsc{GetRandElt}(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}; r_1^{(\alpha)} \cdots r_{f(t)}^{(\alpha)})$;

(c) if $i_{\mathsf{new}} \in \{1, \ldots, \mathsf{c}_{\mathbf{N}}(t)\}$, then:
  i. $\alpha_{i_{\mathsf{new}}} := [N_{t,i_{\mathsf{new}}}]^{\mathsf{A}}(\alpha)$;
  ii. $j_{\alpha_{i_{\mathsf{new}}}} := \textsc{GetIndexOfElt}(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \alpha_{i_{\mathsf{new}}})$;
  iii. $j := j_{\alpha_{i_{\mathsf{new}}}}$;
  iv. output j;

(d) if $i_{\mathsf{new}} = \mathsf{c}_{\mathbf{N}}(t) + 1$, then:
  i. $j_\alpha := \textsc{GetIndexOfElt}(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \alpha)$;
  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) $j_{\mathsf{old}} := Q_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}}), r''_1 \cdots r''_{\mathsf{nr}_{\mathsf{c}}}, i_{\mathsf{new}}\big)$;
   (c) if $j \in \{1, \ldots, 2^{f(t)}\}$, then $j := j_{\mathsf{old}}$;
   (d) if $j - 2^{f(t)} \in \{1, \ldots, \mathsf{length}_{\mathrm{aVRS}}(f(t), f(t), \log|\mathtt{x}|, d, \delta_{\mathsf{c}}, s'_{\mathsf{c}})\}$,
   then $j := \mathsf{length}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}})$
   $+ 2^{f(t)} + \mathsf{length}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}}) + j_{\mathsf{old}}$;
   (e) output j.

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

$$D_{\mathrm{ACSP}}\Big((\mathtt{x}, 1^t), r_1 \cdots r_{\mathsf{rand}_{\mathrm{ACSP}}(\mathtt{x},t)}, \alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{ACSP}}(\mathtt{x},t)}\Big) \equiv$$

1. Parameter instantiation:
   (a) $I_t := \textsc{FindIrrPoly}(1^{f(t)})$;
   (b) $\mathcal{B}_{\mathbb{F}_t} := \textsc{FieldBasis}(I_t)$;
   (c) $(\mathcal{B}_{H_t}, \mathcal{O}_{H_t}) := \textsc{Find}\mathbf{H}(1^t)$;
   (d) for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $[N_{t,i}]^{\mathsf{A}} := \textsc{Find}\mathbf{N}(1^t, i)$;
   (e) $[P_t]^{\mathsf{A}} := \textsc{Find}\mathbf{P}(1^t)$;
   (f) $(\mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}) := \textsc{Find}\mathbf{I}(1^t, 1^{\log|\mathtt{x}|})$;
   (g) $d := \deg(P_t(x, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \ldots, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathsf{c}_{\mathbf{N}}(t)})}))$.
2. Deduce the amount of randomness and number of queries for the amplified verifiers:
   (a) $\delta_{\mathrm{RS}} := (8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$ 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) $\mathsf{nq}_{\mathrm{aVRS}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$;
   (f) $\mathsf{nr}_{\mathrm{aVRS}} := \mathsf{rand}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$;
   (g) $\delta_{\mathsf{c}} := (8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$ and $s'_{\mathsf{c}} := 0.5$;
   (h) $\mathsf{nq}_{\mathsf{c}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \log|\mathtt{x}|, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$;
   (i) $\mathsf{nr}_{\mathsf{c}} := \mathsf{rand}_{\mathrm{aVRS}}(f(t), f(t), \log|\mathtt{x}|, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$.
3. Parse $\alpha_1 \cdots \alpha_{\mathsf{query}_{\mathrm{ACSP}}(\mathtt{x},t)}$ 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. $b_{\mathrm{RS}} := D_{\mathrm{aRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}}), r'_1 \cdots r'_{\mathsf{nr}_{\mathrm{aRS}}}, \alpha'_1 \cdots \alpha'_{\mathsf{nq}_{\mathrm{aRS}}}\big)$.
6. $r''_1 \cdots r''_{\mathsf{nr}_{\mathrm{aVRS}}} := r_1 \cdots r_{\mathsf{nr}_{\mathrm{aVRS}}}$.

7. $b_{\mathrm{VRS}} := D_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{H_t}, \mathcal{O}_{H_t}, d, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}}), r''_1 \cdots r''_{\mathsf{nr}_{\mathrm{aVRS}}}, \alpha''_1 \cdots \alpha''_{\mathsf{nq}_{\mathrm{aVRS}}}\big)$.
8. $r_1^{(\alpha)} \cdots r_{f(t)}^{(\alpha)} := r_1 \cdots r_{f(t)}$;
9. $\alpha := \textsc{GetRandElt}(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}; r_1^{(\alpha)} \cdots r_{f(t)}^{(\alpha)})$.
10. $[P_t]^{\mathsf{A}} := \textsc{Find}\mathbf{P}(1^t)$.
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. $b_{\mathsf{c}} := D_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{I_{t,\log|\mathrm{x}|}}, \mathcal{O}_{I_{t,\log|\mathrm{x}|}}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}}), r'''_1 \cdots r'''_{\mathsf{nr}_{\mathsf{c}}}, \tau_1 \cdots \tau_{\mathsf{nq}_{\mathsf{c}}}\big)$.
15. $b := b_{\mathrm{RS}} \wedge b_{\mathrm{VRS}} \wedge b_{\mathbf{P}} \wedge b_{\mathsf{c}}$.
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 12.21 and Lemma 12.22. □

12.4 Efficient Reverse Sampling

The third property considered by Barak and Goldreich [BG08, 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 12.24 (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_{sACSP} does satisfy the definition above — that this is so is not clear, especially in light of the recursive nature of $V_{\mathrm{RS},=}$, used as part of the construction of V_{sACSP}.

Claim 12.25. *The PCP verifier V_{sACSP} is efficiently reverse samplable.*

Again, the construction of V_{sACSP} is quite complicated, and we will have to proceed one step at a time. As before, our proof will be "bottom up". Please refer to Figure 1 for an algorithmic reference of the construction of V_{sACSP}. (More specific references will be given in each of the proofs.) Also, as we did in Section 12.3, we fix throughout a specific choice of parameters $(\eta, \kappa_0, \gamma, \mu)$, which parametrize V_{sACSP}.

Throughout, the symbol $\circ$ will denote string contatenation.

12.4.1 Efficient Reverse Sampling of $V_{\mathrm{RS},=}$

Lemma 12.26. *The (strong) PCPP verifier $V_{\mathrm{RS},=}$ is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma C.1. We have already established in Lemma 12.16 that $V_{\mathrm{RS},=}$ is a non-adaptive (strong) PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_L, \mathcal{O}_L)$, a query number $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)\}$, and a query index $j \in \{1, \ldots, |L| + \mathsf{length}_{\mathrm{RS},=}(\ell, \kappa)\}$,

$$S_{\mathrm{RS},=}\Big((I_\ell, \mathcal{B}_L, \mathcal{O}_L), i, j\Big) \equiv$$

1. Compute $(r, n, t) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_L, \mathcal{O}_L), i, j\big)$.
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 $\mathcal{B}_L$ and offset $\mathcal{O}_L$ for a κ-dimensional affine subspace $L \subseteq \mathbb{F}_{2^\ell}$, 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 $j = Q_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_L, \mathcal{O}_L), r, i\big)$, 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, $\mathcal{B}_L$, $\mathcal{O}_L$, 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:

$\textsc{RandSamp}\big((I_\ell, \mathcal{B}_L, \mathcal{O}_L), i, j\big) \equiv$

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) $\mathcal{B}_{L_0} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma})$; $\mathcal{O}_{L_0} := \mathcal{O}_L$;
   (d) $\mathcal{B}_{L_0'} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu})$, $\mathcal{O}_{L_0'} := \mathcal{O}_L$;
   (e) $\mathcal{B}_{L_1} := (a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_\kappa)$, $\mathcal{O}_{L_1} := \mathcal{O}_L$;
   (f) $[Z_{L_0}]^{\mathsf{A}} := \textsc{FindSubspPoly}(I_\ell, \mathcal{B}_{L_0}, \mathcal{O}_{L_0})$;
   (g) $\mathcal{B}_{Z_{L_0}(L_1)} := ([Z_{L_0}]^{\mathsf{A}}(a_{\lfloor \kappa/2 \rfloor - \gamma + 1}) + [Z_{L_0}]^{\mathsf{A}}(0_{\mathbb{F}_{2^\ell}}), \ldots, [Z_{L_0}]^{\mathsf{A}}(a_\kappa) + [Z_{L_0}]^{\mathsf{A}}(0_{\mathbb{F}_{2^\ell}}))$;
   (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. $\alpha := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_L, \mathcal{O}_L, j')$;
      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 $\mathcal{B}_{L_\beta} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}, \beta + \mathcal{O}_{L_0'})$;
      v. $\mathcal{O}_{L_\beta} := \mathcal{O}_{L_0'}$;
      vi. if $\alpha \notin L_0'$, then do the following:
         A. $r_1 := 0$;
         B. $r_2 \cdots r_{1+m_0} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L_1}, \mathcal{O}_{L_1}, \beta)$;
         C. $\widetilde{m}_0 := m - 1 - m_0$;

D. if $\widetilde{m} > 0$, draw $r_{m-\widetilde{m}_0+1}, \dots, r_m \in \{0,1\}$ at random;

E. $j'' := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_{L_\beta}, \mathcal{O}_{L_\beta}, \alpha)$;

F. $(\overline{r}_0, n_0, \ell_0) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L_\beta}, \mathcal{O}_{L_\beta}), i, j''\big)$;

G. output $(r_1 \cdots r_m \circ \overline{r}_0, 2^{\widetilde{m}_0} \cdot n_0, 2^m \cdot \ell_0)$;

vii. if $\alpha \in L'_0$, then do the following:

first figure out the probability mass to assign to a row test

A. $r_2^{(0)} \cdots r_{1+m_0}^{(0)} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L_1}, \mathcal{O}_{L_1}, \beta)$;

B. $\widetilde{m}_0 := m - 1 - m_0$;

C. if $\widetilde{m}_0 > 0$, draw $r_{m-\widetilde{m}_0+1}^{(0)}, \dots, r_m^{(0)} \in \{0,1\}$ at random;

D. $j'' := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_{L_\beta}, \mathcal{O}_{L_\beta}, \alpha)$;

E. $(\overline{r}_0, n_0, \ell_0) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L_\beta}, \mathcal{O}_{L_\beta}), i, j''\big)$;

then figure out probability mass to assign to a column test

F. $r_2^{(1)} \cdots r_{1+m_1}^{(1)} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L'_0}, \mathcal{O}_{L'_0}, \alpha)$;

G. $\widetilde{m}_1 := m - 1 - m_1$;

H. if $\widetilde{m}_1 > 0$, draw $r_{m-\widetilde{m}_1+1}^{(1)}, \dots, r_m^{(1)} \in \{0,1\}$ at random;

I. $j'' := \textsc{GetIndexOfElt}(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, \mathcal{O}_{Z_{L_0}(L_1)}, \alpha)$;

J. $(\overline{r}_1, n_1, \ell_1) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L'_0}, \mathcal{O}_{L'_0}), i, j''\big)$;

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 $j' \in \{1, \dots, 2^{\kappa+\mu+1}\}$, 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. $\alpha := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_{L'_0}, \mathcal{O}_{L'_0}, \iota_{\mathsf{col}})$;

iv. $\beta := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, \mathcal{O}_{Z_{L_0}(L_1)} \iota_{\mathsf{row}})$;

first figure out probability mass to assign to a row test

v. $r_2^{(0)} \cdots r_{1+m_0}^{(0)} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L_1}, \mathcal{O}_{L_1}, \beta)$;

vi. $\widetilde{m}_0 := m - 1 - m_0$;

vii. if $\widetilde{m}_0 > 0$, draw $r_{m-\widetilde{m}_0+1}^{(0)}, \dots, r_m^{(0)} \in \{0,1\}$ at random;

viii. $j'' := \iota_{\mathsf{col}}$;

ix. $(\overline{r}_0, n_0, \ell_0) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L_\beta}, \mathcal{O}_{L_\beta}), i, j''\big)$;

then figure out probability mass to assign to a column test

x. $r_2^{(1)} \cdots r_{1+m_1}^{(1)} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L'_0}, \mathcal{O}_{L'_0}, \alpha)$;

xi. $\widetilde{m}_1 := m - 1 - m_1$;

xii. if $\widetilde{m}_1 > 0$, draw $r_{m-\widetilde{m}_1+1}^{(1)}, \dots, r_m^{(1)} \in \{0,1\}$ at random;

xiii. $j'' := \iota_{\mathsf{row}}$;

xiv. $(\overline{r}_1, n_1, \ell_1) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L'_0}, \mathcal{O}_{L'_0}), i, j''\big)$;

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 $j' \in \{1, \dots, 2^{\lceil \kappa/2 \rceil + \gamma}\}$, 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. $\beta' := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, \mathcal{O}_{Z_{L_0}(L_1)}, j')$;
iii. deduce $\beta \in L_1$ such that $[Z_{L_0}]^{\mathsf{A}}(\beta) = \beta'$;
iv. $r_2 \cdots r_{1+m_0} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L_1}, \mathcal{O}_{L_1}, \beta)$;
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. $(\overline{r}_0, n_0, \ell_0) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{L_\beta}, \mathcal{O}_{L_\beta}), i, j''\big)$;
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. $\alpha := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_{L'_0}, \mathcal{O}_{L'_0}, j')$;
iii. deduce $\beta \in L_1$ such that $[Z_{L_0}]^{\mathsf{A}}(\beta) = [Z_{L_0}]^{\mathsf{A}}(\alpha)$;
iv. $r_2 \cdots r_{1+m_1} := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_{L'_0}, \mathcal{O}_{L'_0}, \alpha)$;
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. $(\overline{r}_1, n_1, \ell_1) := \textsc{RandSamp}\big((I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, \mathcal{O}_{Z_{L_0}(L_1)}), i, j''\big)$;
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 C.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 the unique $\beta \in L_1$ 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} (see Algorithm 21) also implies a "sparse" computation for finding β. Specifically, because Z_{L_0} is a linearized polynomial (or, more precisely, a shift of such a polynomial), the map $Z_{L_0} \colon \mathbb{F}_{2^\ell} \to \mathbb{F}_{2^\ell}$ is an affine map. Hence, finding β reduces to finding the unique solution in L_1 of a $\dim(L_0)$-dimensional linear system $Mx + \mathcal{O} = \beta'$ where M and $\mathcal{O}$ are the matrix and offset induced by the affine 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.

12.4.2 Efficient Reverse Sampling of $V_{\mathrm{RS},<}$

Lemma 12.27. *The (strong) PCPP verifier $V_{\mathrm{RS},<}$ is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma C.2. We have already established in Lemma 12.17 that $V_{\mathrm{RS},<}$ is a non-adaptive (strong) PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d)$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},<}(\ell, \kappa, d)\}$, and $j \in \{1, \ldots, |S| + \mathsf{length}_{\mathrm{RS},<}(\ell, \kappa, d)\}$,

$$S_{\mathrm{RS},<}\Big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d), i, j\Big) \equiv$$

1. If $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},=}(\ell, \kappa)\}$, then:
   (a) compute $r := S_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S), i, j\big)$.
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 $r := S_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S), i', j\big)$;
   (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 $r := S_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S), i', j'\big)$.
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 12.16. □

12.4.3 Efficient Reverse Sampling of $V_{\mathrm{RS},>}$

Lemma 12.28. *The (strong) PCPP verifier $V_{\mathrm{RS},>}$ is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma C.3. We have already established in Lemma 12.18 that $V_{\mathrm{RS},>}$ is a non-adaptive (strong) PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d)$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS},>}(\ell, \kappa, d)\}$, and $j \in \{1, \ldots, |S| + \mathsf{length}_{\mathrm{RS},>}(\ell, \kappa, d)\}$,

$$S_{\mathrm{RS},>}\Big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d), i, j\Big) \equiv$$

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. $r := S_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$;
      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. $r := S_{\mathrm{RS},<}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d_{\kappa,\eta,d}), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$;
      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. $\gamma := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, j_{\mathsf{new}})$;

      C. $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \gamma)$;

      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. $\tau := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, j_{\mathsf{new}})$;

      C. $\gamma := \tau^{1/((i_{\mathsf{new}}-2)\cdot(d_{\kappa,\eta}+1))}$;

      D. $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \gamma)$;

      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. $\gamma := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, j_{\mathsf{new}})$;

      C. $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \gamma)$;

      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. $\tau := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, j_{\mathsf{new}})$;

      C. $\gamma := \tau^{1/((i_{\mathsf{new}}-2)\cdot(d_{\kappa,\eta}+1))}$;

      D. $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \gamma)$;

      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 12.16 and Lemma 12.17. □

12.4.4 Efficient Reverse Sampling of V_{RS}

Lemma 12.29. *The (strong) PCPP verifier V_{RS} is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma C.4. We have already established in Lemma 12.19 that V_{RS} is a non-adaptive (strong) PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d)$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d)\}$, and $j \in \{1, \ldots, |S| + \mathsf{length}_{\mathrm{RS}}(\ell, \kappa, d)\}$,

$$S_{\mathrm{RS}}\Big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d), i, j\Big) \equiv$$

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 $r := S_{\mathrm{RS},<}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d), i, j\big)$.
   (c) If $d = d_{\kappa,\eta}$, then output $r := S_{\mathrm{RS},=}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S), i, j\big)$.
   (d) If $d > d_{\kappa,\eta}$, then output $r := S_{\mathrm{RS},>}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d), i, j\big)$.
   (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 12.16, Lemma 12.17, and Lemma 12.18. □

12.4.5 Efficient Reverse Sampling of V_{VRS}

Lemma 12.30. *The (strong) PCPP verifier V_{VRS} is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma C.5. We have already established in Lemma 12.20 that V_{VRS} is a non-adaptive (strong) PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \mathcal{B}_H, \mathcal{O}_H, d)$, $i \in \{1, \dots, \mathsf{query}_{\mathrm{VRS}}(\ell, \kappa, \lambda, d)\}$, and $j \in \{1, \dots, |S| + \mathsf{length}_{\mathrm{VRS}}(\ell, \kappa, \lambda, d)\}$,

$$S_{\mathrm{VRS}}\Big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, \mathcal{B}_H, \mathcal{O}_H, d), i, j\Big) \equiv$$

1. If $i \in \big\{1, \dots, \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d - |H|)\big\}$, then:
   (it must be that $j \in \{|S|+1, \dots, |S| + |S| + \mathsf{length}_{\mathrm{RS}}(\ell, \kappa, d - |H|)\}$)
   (a) $i_{\mathsf{new}} := i$;
   (b) $j_{\mathsf{new}} := j - |S|$;
   (c) $r := S_{\mathrm{RS}}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d - |H|), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$.
2. If $i = \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d - |H|) + 1$, then:
   (it must be that $j \in \{1, \dots, |S|\}$)
   (a) $j_{\mathsf{new}} := j$;
   (b) $\alpha := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, j_{\mathsf{new}})$;
   (c) $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \alpha)$.
3. If $i = \mathsf{query}_{\mathrm{RS}}(\ell, \kappa, d - |H|) + 2$, then:
   (it must be that $j \in \{|S|+1, \dots, |S| + |S|\}$)
   (a) $j_{\mathsf{new}} := j - |S|$;
   (b) $\alpha := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, j_{\mathsf{new}})$;
   (c) $r := \textsc{GetRandFromElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \alpha)$.
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 12.19. □

12.4.6 Efficient Reverse Sampling of V_{aRS}

Lemma 12.31. *The PCPP verifier V_{aRS} is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma C.6. We have already established in Lemma 12.21 that V_{aRS} is a non-adaptive PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d, \delta, s')$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{aRS}}(\ell, \kappa, d, \delta, s')\}$, and $j \in \{1, \ldots, |S| + \mathsf{length}_{\mathrm{aRS}}(\ell, \kappa, d, \delta, s')\}$,

$$S_{\mathrm{aRS}}\Big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d, \delta, s'), i, j\Big) \equiv$$

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. $r_z := S_{\mathrm{RS}}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, d), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$.
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 12.19. □

12.4.7 Efficient Reverse Sampling of V_{aVRS}

Lemma 12.32. *The PCPP verifier V_{aVRS} is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma C.7. We have already established in Lemma 12.22 that V_{aVRS} is a non-adaptive PCPP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \mathcal{B}_H, \mathcal{O}_H, d, \delta, s')$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{aVRS}}(\ell, \kappa, \lambda, d, \delta, s')\}$, and $j \in \{1, \ldots, |S| + \mathsf{length}_{\mathrm{aVRS}}(\ell, \kappa, \lambda, d, \delta, s')\}$,

$$S_{\mathrm{aVRS}}\Big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, \mathcal{B}_H, \mathcal{O}_H, d, s'), i, j\Big) \equiv$$

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. $r_z := S_{\mathrm{VRS}}\big((I_\ell, \mathcal{B}_S, \mathcal{O}_S, \mathcal{B}_H, \mathcal{O}_H, d), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$.
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 12.20. □

12.4.8 Efficient Reverse Sampling of V_{sACSP}

Lemma 12.33. *The PCP verifier V_{sACSP} is efficiently reverse-samplable.*

Proof. Recall the definitions from Lemma C.8. We have already established in Lemma 12.23 that V_{aVRS} is a non-adaptive PCP verifier. We are left to construct the probabilistic polynomial-time "reverse sampler": on input $(\mathtt{x}, 1^t)$, $i \in \{1, \ldots, \mathsf{query}_{\mathrm{ACSP}}(\mathtt{x}, t)\}$, and $j \in \{1, \ldots, \mathsf{length}_{\mathrm{ACSP}}(\mathtt{x}, t)\}$,

$$S_{\mathrm{ACSP}}\Big((\mathtt{x}, 1^t), i, j\Big) \equiv$$

1. Parameter instantiation:
   (a) $I_t := \textsc{FindIrrPoly}(1^{f(t)})$;
   (b) $\mathcal{B}_{\mathbb{F}_t} := \textsc{FieldBasis}(I_t)$;
   (c) $(\mathcal{B}_{H_t}, \mathcal{O}_{H_t}) := \textsc{Find}\mathbf{H}(1^t)$;
   (d) for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $[N_{t,i}]^{\mathsf{A}} := \textsc{Find}\mathbf{N}(1^t, i)$;
   (e) $[P_t]^{\mathsf{A}} := \textsc{Find}\mathbf{P}(1^t)$;
   (f) $(\mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}) := \textsc{Find}\mathbf{I}(1^t, 1^{\log|\mathtt{x}|})$;
   (g) $d := \deg(P_t(x, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \ldots, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathsf{c}_{\mathbf{N}}(t)})}))$.
2. Deduce the number of queries for the amplified verifiers:
   (a) $\delta_{\mathrm{RS}} := (8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$ 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) $\mathsf{nq}_{\mathrm{aVRS}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$;
   (e) $\delta_{\mathsf{c}} := (8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$ and $s'_{\mathsf{c}} := 0.5$;
   (f) $\mathsf{nq}_{\mathsf{c}} := \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \log|\mathtt{x}|, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$.
3. If $i \in \{1, \ldots, \mathsf{nq}_{\mathrm{aRS}}\}$, then:
   (a) $i_{\mathsf{new}} := i$;
   (b) $j_{\mathsf{new}} := j$;
   (c) $r := S_{\mathrm{aRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}}), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$.
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) $r := S_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{H_t}, \mathcal{O}_{H_t}, d, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}}), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$.
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. $\alpha_{i_{\mathsf{new}}} := \textsc{GetEltWithIndex}(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, j_{\mathsf{new}})$;
      iii. find a random solution α to the equation $\alpha_{i_{\mathsf{new}}} := [N_{t,i_{\mathsf{new}}}]^{\mathsf{A}}(x)$;
      iv. $r := \textsc{GetRandFromElt}(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \alpha)$;
   (b) if $i_{\mathsf{new}} = \mathsf{c}_{\mathbf{N}}(t) + 1$, then:
      i. $j_{\mathsf{new}} := j$;
      ii. $\alpha := \textsc{GetEltWithIndex}(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, j_{\mathsf{new}})$;
      iii. $r := \textsc{GetRandFromElt}(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \alpha)$.
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 $r := S_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}, \delta_{\mathsf{c}}, s'_{\mathsf{c}}), i_{\mathsf{new}}, j_{\mathsf{new}}\big)$;
   (b) if $j' \in \{2^{f(t)} + 1, \ldots, 2^{f(t)} + \mathsf{length}_{\mathrm{aVRS}}(f(t), f(t), \log|\mathtt{x}|, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})\}$,
      where $j' := j - \mathsf{length}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathrm{RS}}, s'_{\mathrm{RS}})$
      $-2^{f(t)} - \mathsf{length}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \delta_{\mathrm{VRS}}, s'_{\mathrm{VRS}})$,
      then:
      i. compute $r := S_{\mathrm{aVRS}}\big((I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}, \delta_{\mathsf{c}}, s'_{\mathsf{c}}), i_{\mathsf{new}}, j'\big)$;
7. Pad r with random bits until it is $\mathsf{rand}_{\mathrm{ACSP}}(\mathtt{x}, t)$-bits long.
8. Output r.

The correctness of S_{ACSP} easily follows by inspection, as well as from Lemma 12.21 and Lemma 12.22.

Note that in Step 5(a)iii we have to find a random root of the polynomial $N_{t,i_{\mathsf{new}}}(x) - \alpha_{i_{\mathsf{new}}}$ over $\mathbb{F}_{2^{f(t)}}$. This can be done in time $\mathsf{poly}(\deg(N_{t,i_{\mathsf{new}}}), f(t))$ by factoring the polynomial (e.g., see the survey of [VZGP01]) and selecting one of the roots at random. $\square$

12.5 Proof Of Knowledge

The fourth property considered by Barak and Goldreich [BG08, Definiton 3.2, third item] for a PCP verifier is a proof-of-knowledge property. We show that the PCP verifier V_{sACSP} 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 12.34 (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 $i = 1, \ldots, T$, $\Pr[E^{\pi}_{\text{PCP}}(x, i) = w_i] > 2/3$.*

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 12.35 (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 $\Pr[E^{\pi}_{\text{PCP}}(x, 1^T) = w] > 2/3$.*

We prove that V_{sACSP} satisfies this weaker definition (and then in Remark 12.37 we explain why it does *not* satisfy the original one). Later, in Section 12.6, 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 12.36. *The PCP verifier V_{sACSP} has the PCP explicit proof-of-knowledge property with error $\varepsilon(n) = 1/2$.*

Proof. The PCP verifier V_{sACSP} is described in Construction 8.5 (and then again more succinctly in Algorithm 9). The PCP weak proof-of-knowledge property of V_{sACSP} can be deduced by examining the part of the proof of Theorem 8.1 that establishes the soundness property of V_{sACSP}. Details follow.

Fix an instance $(\mathtt{x}, 1^t)$ and a proof $\pi = (p_0, \pi_1, p_1, \pi_1, \pi_{\mathtt{c}})$. Suppose that $\Pr\big[V^{\pi}_{\text{sACSP}}\big((\mathtt{x}, 1^t)\big) = 1\big] > 1/2$. Let $I_{t,\log|\mathtt{x}|}$ be the $(\log|\mathtt{x}|)$-th affine subspace in $\vec{I}_t$, $A_{\mathtt{x}}$ be the low-degree extension of the function $f_{\mathtt{x}} \colon I_{t,\log|\mathtt{x}|} \to \{0,1\}$ defined by $f_{\mathtt{x}}(\alpha_i) := \mathtt{x}_i$ where α_i is the i-th element in $I_{t,\log|\mathtt{x}|}$, and $p_{\mathtt{x}}$ the evaluation of $A_{\mathtt{x}}$ on $\mathbb{F}_t$. Also let $d := \deg(P_t(x, x^{(2^{\mathtt{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \ldots, x^{(2^{\mathtt{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathtt{c}_{\mathbf{N}}(t)})}))$.

It must be that

- p_0 is δ_{RS}-close to $\mathrm{RS}(\mathbb{F}_t, \mathbb{F}_t, 2^{\mathtt{m}_{\mathbf{H}}(t)} - 1)$,
- p_1 is δ_{VRS}-close to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, H_t, d)$, and

- $p_0 - p_{\mathtt{x}}$ is δ_{c}-close to $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, I_{t,\log|\mathtt{x}|}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$,

where $\delta_{\mathrm{RS}} = \delta_{\mathsf{c}} = (8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$ and $\delta_{\mathrm{VRS}} = \frac{1}{8}$. Indeed, if that were not the case, then $V^{\pi}_{\mathrm{sACSP}}\big((\mathtt{x}, 1^t)\big)$ 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 $2\delta_{\mathrm{RS}} = (4\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$. (See Remark 8.6 and Lemma 4.11.) Furthermore, we can deduce that A is consistent with the instance $(\mathtt{x}, 1^t)$: $p_0 - p_{\mathtt{x}}$ is $(8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$-close to the evaluation of $A - A_{\mathtt{x}}$ on $\mathbb{F}_t$ and $(8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$-close the code $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, I_{t,\log|\mathtt{x}|}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$; thus, invoking once again the fact that $1 - (2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)/|\mathbb{F}_t|$ is larger than $(4\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$ (and Lemma 4.11) we deduce that the evaluation of $A - A_{\mathtt{x}}$ is in $\mathrm{VRS}(\mathbb{F}_t, \mathbb{F}_t, I_{t,\log|\mathtt{x}|}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1)$.

Let $B\colon \mathbb{F}_t \to \mathbb{F}_t$ be the polynomial defined as $B(x) := P_t\big(x, A(N_{t,1}(x)), \ldots, A(N_{t,\mathsf{c}_{\mathbf{N}}(t)}(x))\big)$.

Because p_0 is $(8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$-close to the evaluation table of A over $\mathbb{F}_t$, we deduce that, for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $p_0 \circ N_{t,i}$ is $\frac{\deg(N_{t,i})}{8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i})}$-close to the evaluation table of $A \circ N_{t,i}$.

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

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

We deduce then, via a union bound, 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 the unique polynomial of degree at most d vanishing on H_t whose evaluation table over $\mathbb{F}_t$ is closest to p_1. Again note that B' is unique, because $1 - d/|\mathbb{F}_t|$ is larger than $2\delta_{\mathrm{VRS}} = 1/4$. (See Remark 8.6 and Lemma 4.11.) 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 $d \leq |\mathbb{F}_t|/4$, 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_{sACSP} accepts with probability at most $1/8 + 1/8 + 1/4 \leq 1/2$ — a contradiction to the fact that $V^{\pi}_{\mathrm{sACSP}}((\mathtt{x}, 1^t))$ 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 $(\mathtt{x}, 1^t)$) we deduce that A is a witness to the fact that $(\mathtt{x}, 1^t) \in$ sACSP.

In summary, p_0 a function over $\mathbb{F}_t$ that is $(8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$-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 $(\mathtt{x}, 1^t) \in$ sACSP. Therefore, we can let E_{PCP} be the algorithm that decodes the Reed–Solomon codeword p_0. More precisely, by Claim 4.12, 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, which is the case in this proof). We can then define E_{PCP} as follows:

$E^{\pi}_{\mathrm{PCP}}\big(\mathtt{x}, 1^{2^t}\big) \stackrel{\mathrm{def}}{=}$

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_{sACSP}, the knowledge extractor E_{PCP} also knows the choice of parameters for sACSP; so, for example, it knows the functions f and $\mathsf{m}_{\mathbf{H}}(t)$. Clearly, E_{PCP} runs in polynomial time.[30]

Thus, it is easy to see that whenever V_{sACSP} accepts the instance $(\mathtt{x}, 1^t)$ 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.) $\square$

Remark 12.37. 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_{sACSP} on input $(\mathtt{x}, 1^t)$ and a witness A for $(\mathtt{x}, 1^t)$), 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_{sACSP} 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 12.6, for many applications of universal arguments this is not an issue.

Remark 12.38. 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 12.39. Barak and Goldreich claimed [BG08, Appendix A] that the PCP proof of knowledge (see Definition 12.34) holds for many PCPs, yet the PCPs that we use only satisfy the explicit PCP proof of knowledge (see Definition 12.35). The reason is that many PCP constructions (e.g., [BFLS91]) use *multivariate* techniques, as opposed to *univariate* techniques as we do. More precisely, many 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].

As discussed before, the advantage of univariate techniques over multivariate techniques is that the proof length of the PCP oracle can be made very short; namely, we do not know of PCPs of quasilinear length based on multivariate techniques but we do when based on univariate techniques.

12.6 Obtaining (Almost) Universal Arguments

Having proved that the PCP system $(P_{\text{sACSP}}, V_{\text{sACSP}})$ for sACSP does satisfy the first three properties and a variant of the fourth property considered by Barak and Goldreich [BG08, Definiton 3.2], we now explain how these four properties are enough to imply, through [BG08, Construction 3.4], an *almost universal argument system* for sACSP. (See Definition 12.9.)

[30] 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 sACSP problems is $2^{\mathsf{m}_{\mathbf{H}}(t)}$ and not 2^t.

Claim 12.40. *Following [BG08, Construction 3.4] by starting with the (amplified) PCP system* $(P_{\mathrm{sACSP}}, V_{\mathrm{sACSP}})$ *and a collision-resistant hash function yields an almost universal argument system* $(P_{\mathsf{UA}}, V_{\mathsf{UA}})$ *for the language* sACSP.

We assume familiarity with the proof of [BG08, Lemma 3.5], which shows that the construction of universal arguments given in [BG08, Construction 3.4] using a PCP system satisfying the *original* four properties in [BG08, Definition 3.2] works, as long as a collision-resistant function family is used.

In comparison, our Claim 12.40 says that, even if we start with a PCP system that satisfies a slightly weaker notion of proof of knowledge (i.e., Definition 12.8 instead of Definition 12.6), the construction of universal arguments still yields a universal argument, albeit with a correspondingly weaker notion of proof of knowledge, that is, yields 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 [BG08, Construction 3.4].)

As discussed in Remark 12.2, this is sufficient for most "positive" applications, because an almost universal argument is, in particular, also a succinct argument of knowledge for NP, and this latter primitive is usually sufficient for applications.

Proof of Claim 12.40. We give a high-level overview of the proof [BG08, 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_{sACSP} has been amplified to error $2^{-\kappa}$ for inputs of length κ; in particular, it has the PCP explicit proof-of-knowledge property with error $\varepsilon(\kappa) = 2^{-\kappa}$. (See Remark 12.38.)

Roughly, the argument used to prove [BG08, 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). This step crucially relies on the existence of an efficient reverse sampler.

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 12.6.

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 12.8 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 $(\mathtt{x}, 1^t) \in \{0,1\}^\kappa$, if $\widetilde{P}_{\mathsf{UA},\kappa}$ convinces V_{UA} to accept $(\mathtt{x}, 1^t)$ 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 $(\mathtt{x}, 1^t)$, outputs a valid witness $\mathtt{w}$ for $\mathtt{x}$.

Barak and Goldreich [BG08, 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 $(\mathtt{x}, 1^t)$ with probability greater than $p_{\mathsf{UA}}(\kappa)^{-1}$, then, with probability $\frac{1}{4.5 p_{\mathsf{UA}}(\kappa)}$, $\mathsf{recover}^{\langle \widetilde{P}_{\mathsf{UA},\kappa}\rangle}\big((\mathtt{x}, 1^t), \cdot\big)$ is an implicit representation of a PCP oracle π such that $\Pr\left[V^{\pi}_{\mathrm{sACSP}}\big((\mathtt{x}, 1^t)\big) = 1\right] > \frac{1}{8 p_{\mathsf{UA}}(\kappa)} > \frac{1}{2^\kappa}$.[31]

Also, we amplify the success probability of $E_{\mathrm{PCP}}(\mathtt{x}, 1^{2^t})$, with $(\mathtt{x}, 1^t) \in \{0,1\}^\kappa$, 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 $(\mathtt{x}, 1^t) \in \{0,1\}^\kappa$,

$E_{\mathsf{UA}}^{\langle \widetilde{P}_{\mathsf{UA},\kappa}\rangle}(\mathtt{x}, 1^{2^t}) \stackrel{\text{def}}{=}$

1. Draw a random tape ω for the oracle-recovery procedure $\mathsf{recover}$.
2. Invoke $E_{\mathrm{PCP}}(\mathtt{x}, 1^{2^t})$ providing it with oracle access to π, using the oracle-recovery procedure $\mathsf{recover}$ on input $(\mathtt{x}, t)$, random tape ω, and oracle access to the code of $\widetilde{P}_{\mathsf{UA},\kappa}$.
3. Output whatever (the amplified) $E^{\pi}_{\mathrm{PCP}}(\mathtt{x}, 1^{2^t})$ 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 $\mathsf{poly}(|y|, 2^t, \kappa)$ invocations of $\mathsf{recover}_\varepsilon$ (as the amplified E_{PCP} runs in $\mathsf{poly}(|y|, 2^t, \kappa)$ time).

Finally, the probability that $E_{\mathsf{UA}}^{\langle \widetilde{P}_{\mathsf{UA},\kappa}\rangle}(\mathtt{x}, 1^{2^t})$ succeeds in extracting a valid witness for y is at least the probability that ω is a "good" random tape for $\mathsf{recover}\big((\mathtt{x}, t)\big)$ (i.e., makes it an implicit representation of a "convincing" PCP oracle π) times the probability that $E^{\pi}_{\mathrm{PCP}}(\mathtt{x}, 1^{2^t})$ successfully extracts; for sufficiently large $\kappa \in \mathbb{N}$, that probability is at least $\frac{1}{4.5 p_{\mathsf{UA}}(\kappa)} \cdot (1 - 2^{-\kappa}) > \frac{1}{5 p_{\mathsf{UA}}(\kappa)}$, as desired. □

Remark 12.41. Note that the alphabet of the PCP oracle is elements of a finite field $\mathbb{F}_{2^\ell}$, and not bits. Fortunately, 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 require any changes in the proof.

[31] 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.

A Work On Fast Verification Of Long Computations

While we believe that the question of whether PCPs can be made efficient enough to be used in practice (e.g., via the "commit to a PCP and then reveal" construction [Kil92, Mic00, BG08, DCL08, BCCT12a, DFH12, GLR11]) is itself fascinating, a natural question to consider is whether the heavy machinery of PCPs is really needed for the application of fast verification of long computations.

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.

At present, we know that:

- Ishai et al. [IKO07] constructed a four-message argument system for NP in which the prover-to-verifier communication is short (i.e., an argument with a *laconic* prover [GVW02]) by combining a *strong linear PCP* and (standard) linear homomorphic encryption; they also showed how to extend their approach to "balance" the communication between the prover and verifier and obtain a $O(1/\varepsilon)$-message argument system for NP with $O(n^{\varepsilon})$ communication complexity.

  While their basic protocol does not provide the verifier with any saving in computation, Ishai et al. noted that their protocol actually yields a *batching argument*: namely, an argument in which, in order to simultaneously verify the correct evaluation of ℓ circuits of size S, the verifier may run in time S (i.e., in time S/ℓ per circuit evaluation). A set of works [SBW11, SMBW12, SVP$^+$12, SBV$^+$12] has improved upon, optimized, and implemented the batching argument of Ishai et al. [IKO07] for the purpose of verifiable delegation of computation.

- Goldwasser et al. [GKR08] showed how to verify log-space uniform NC computations (i.e., log-space uniform circuits of polylogarithmic depth) by relying on probabilistic-checking techniques that are lighter than those used in PCPs; their protocol can be reduced to one round [KR09, KRR12]. A set of works [CMT12, TRMP12] has optimized and implemented the protocol of Goldwasser et al. [GKR08]. Canetti et al. [CRR11] showed how to extend the techniques in [GKR08] to also handle non-uniform NC circuits, in a publicly-verifiable one-round protocol having an expensive offline preprocessing phase.

- A set of works [GGP10, CKV10, AIK10] showed how to verify all polynomial-time computations without using probabilistic-checking techniques, provided one is willing to tolerate an expensive offline preprocessing phase.[32] These works, however, 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.

- Groth [Gro10] showed how to construct publicly-verifiable SNARGs of knowledge (SNARKs) for all of NP with an expensive offline preprocessing phase and where both the preprocessing running time and the prover running are quadratic (in the size of the circuit verifying membership in the language). Lipmaa [Lip12] improved on the work of Groth by showing how to make the preprocessing running time quasilinear; however, the prover running time remained quadratic in his work. Gennaro et al. [GGPR12] obtained quasi-optimal time complexities

[32] Though, admittedly, these works rely on fully-homomorphic encryption [Gen09], which, just like PCPs, is also notoriously inefficient in practice.

in this setting: both the prover running time and preprocessing running time are quasilinear in their work.

Bitansky et al. [BCI$^+$13] showed how to compile a *weak linear* PCP to a SNARK with an expensive offline preprocessing phase. Constructions of linear PCPs for NP can easily be obtained, for example, via probabilistic-checking techniques for the Hadamard code [ALM$^+$98] or via quadratic-span programs [GGPR12]. While none of [Gro10, Lip12, GGPR12] explicitly invoke probabilistic-checking techniques, the result of Bitansky et al. shows that all these constructions can be interpreted as implicitly invoking the power of weak linear PCPs, thereby providing a unifying explanation of the constructions in these works.

All of [Gro10, Lip12, GGPR12, BCI$^+$13] rely on various flavors of knowledge-of-exponent assumptions [Dam92, HT98, BP04] in bilinear groups or, more generally, on the existence of encodings that *only* allow for linear homomorphic operations [BCI$^+$13]; relying on *non-falsifiable* assumptions [Nao03] such as the aforementioned ones seems inherent for obtaining SNARGs [GW11].

- Bitansky et al. [BCCT12b] showed that the expensive preprocessing phase of a SNARK (if there is one) can always be removed via a generic transformation that only relies on standard cryptographic assumptions and does not invoke any probabilistic-checking techniques. Their transformation also shows how to make the prover complexity tightly related in time *and space* to those of the original computation to be verified.
- Bitansky and Chiesa [BC12] showed how to use multi-prover interactive proof techniques [BOGKW88] to obtain, from standard cryptographic assumptions, interactive succinct arguments where again the prover's time and space complexities are tightly related to those of the original computation to be verified; they also showed how to obtain designated-verifier SNARKs, where the prover has similar time and space complexities, from a knowledge assumption about fully-homomorphic encryption.

Thus, it seems that the question of whether full-fledged PCPs are really needed depends on the strength of the desired application (e.g., verify deterministic vs. non-deterministic computations, publicly-verifiable vs. privately-verifiable, interactive vs. non-interactive) and the strength of the computational assumptions we are willing to make (e.g., only falsifiable assumptions or not).

For example, on the one hand, we could make strong non-falsifiable assumptions to only use linear PCPs [BCI$^+$13] and then invoke the result of [BCCT12b] 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, Mic00, BG08]. Finally, we note that the use of PCPs in constructing succinct arguments is to a certain extent inherent [RV09].

B PCP Algorithmic Specifications

We provide *algorithmic specifications* for the PCP system used in the proof of Theorem 1.

The probabilistic-checking core of the PCP system used in the proof of Theorem 1 is the PCP system $(P_{\text{sACSP}}, V_{\text{sACSP}})$ for the language sACSP used in the proof of Theorem 8.1 in Section 8; the remaining part of the construction consists instead of Levin reductions.

Recall that a PCP system for the language sACSP would indeed not be very useful without sufficiently tight Levin reductions from "more natural" languages. Such reductions are studied in detail by Ben-Sasson et al. [BSCGT13], who consider Levin reductions from the correctness of computation of a wide class of random-access machines.

In the algorithmic specifications of this section, we shall thus ignore the details of the specific Levin reduction to sACSP, and only focus on spelling out the details of $(P_{\text{sACSP}}, V_{\text{sACSP}})$; a complexity analysis of the PCP system's proof length, randomness, and query complexity is given later in Appendix C.

Concretely, we assume that, for some machine M of interest, we *already* know a Levin reduction $(\mathsf{Params}_L, \mathsf{Reduce\text{-}Witness}_L, \mathsf{Recover\text{-}Witness}_L)$ from the language

$$L = \left\{ (\mathtt{x}, 1^t) : \text{ there is } \mathtt{w} \text{ with } |\mathtt{w}| \leq 2^t \text{ s.t. } M(\mathtt{x}, \mathtt{w}) \text{ accepts within } 2^t \text{ steps} \right\}$$

to the language sACSP with the choice of parameters Params_L. That is, $\mathsf{Reduce\text{-}Witness}_L$ and $\mathsf{Recover\text{-}Witness}_L$ are polynomial-time computable and the following two properties hold:

1. **Soundness:** $(\mathtt{x}, 1^t) \in L \longleftrightarrow (\mathtt{x}, 1^t) \in \text{sACSP}(\mathsf{Params}_L)$
2. **Witness Reductions:**
   - $\begin{array}{c}\mathtt{w} \text{ is witness to} \\ (\mathtt{x}, 1^t) \in L\end{array} \longrightarrow \begin{array}{c}\mathsf{Reduce\text{-}Witness}_L(\mathtt{w}) \text{ is witness to} \\ (\mathtt{x}, 1^t) \in \text{sACSP}(\mathsf{Params}_L)\end{array}$
   - $\begin{array}{c}\mathsf{Recover\text{-}Witness}_L(A) \text{ is witness to} \\ (\mathtt{x}, 1^t) \in L\end{array} \longleftarrow \begin{array}{c}A \text{ is witness to} \\ (\mathtt{x}, 1^t) \in \text{sACSP}(\mathsf{Params}_L)\end{array}$

Note that the input $\mathtt{x}$ itself is not changed by the Levin reduction considered here (i.e., L is reducible to sACSP with the choice of parameters Params_L via the identity mapping).[33]

In Figure 1 we give a high-level diagram of all the components (together with the dependencies among them) of the PCP system (P_L, V_L) for the language L. Following are pseudocode listings for each component: in Section B.1 are listings for the components of the prover P_L, in Section B.2 are listings for the components of the verifier V_L, and finally in Section B.3 are listings for the finite-field algorithms used throughout.

[33] Recall from Definition 7.1 that sACSP only considers $(\mathtt{x}, 1^t)$ such that $|\mathtt{x}| \in \{2, \ldots, 2^t\}$.

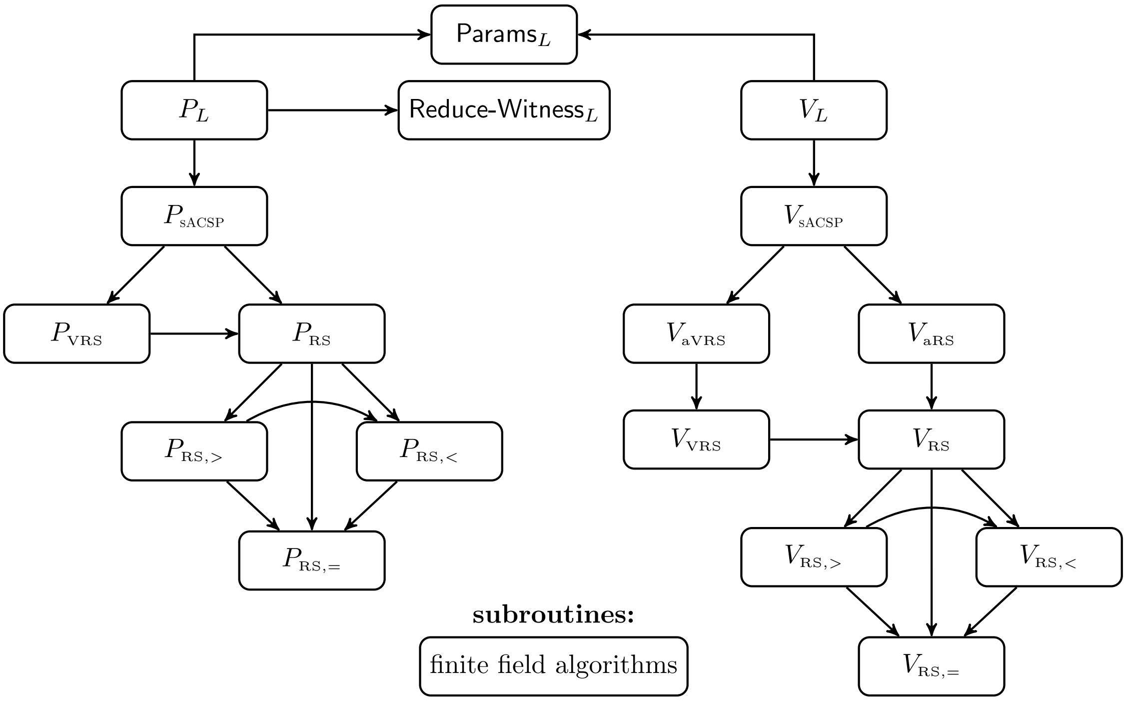

Figure 1: A high-level diagram showing the relation among the "modules" in the construction of the PCP prover P_L and PCP verifier V_L. Also, every module makes use of certain algorithms for computations in finite fields, which we spell out.

B.1 PCP Prover Specifications

Algorithm 1 P_L

---

inputs:
- $(\mathtt{x}, 1^t)$, in the language L
- $\mathtt{w}$, witness to $(\mathtt{x}, 1^t) \in L$

randomness:
- no randomness used

output:
- π, probabilistically-checkable proof for the satisfiability of $(\mathtt{x}, 1^t)$

1. $A \leftarrow \mathsf{Reduce\text{-}Witness}_L(\mathtt{w})$
2. $\pi \leftarrow P_{\text{sACSP}}\big((\mathtt{x}, 1^t), A\big)$
3. output π

notes:
- see Algorithm 8 for V_L, the corresponding verifier
- P_{sACSP} has the choice of parameters Params_L hardcoded

Algorithm 2 P_{sACSP}

inputs:
- $(\mathtt{x}, 1^t)$, in sACSP with the given choice of parameters

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

- $A \colon \mathbb{F}_t \to \mathbb{F}_t$, assignment polynomial that is witness to $(\mathtt{x}, t) \in$ sACSP
(i.e., $\deg(A) < 2^{\mathsf{m}_{\mathbf{H}}(t)}$, A satisfies the constraint polynomial P_t, and A is consistent with $(\mathtt{x}, 1^t)$, cf. Definition 7.1)

randomness:
- no randomness used

output:
- $\pi = (p_0, \pi_0, p_1, \pi_1, \pi_{\mathsf{c}})$, probabilistically-checkable proof for the claim "$(\mathtt{x}, 1^t) \in$ sACSP"

1. Parameter instantiation:
   (a) $I_t := \textsc{FindIrrPoly}(1^{f(t)})$
   (b) $\mathcal{B}_{\mathbb{F}_t} := \textsc{FieldBasis}(I_t)$
   (c) $(\mathcal{B}_{H_t}, \mathcal{O}_{H_t}) := \textsc{Find}\mathbf{H}(1^t)$
   (d) for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $[N_{t,i}]^{\mathsf{A}} := \textsc{Find}\mathbf{N}(1^t, i)$
   (e) $[P_t]^{\mathsf{A}} := \textsc{Find}\mathbf{P}(1^t)$
   (f) $(\mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}) := \textsc{Find}\mathbf{I}(1^t, 1^{\log|\mathtt{x}|})$
2. Proof that A has low degree:
   (a) $p_0 := \textsc{SubspaceEval}(I_\ell, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, A)$
   (b) $\pi_0 := P_{\text{RS}}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, A\big)$
3. Proof that A satisfies the constraint polynomial:
   (a) for each $\alpha \in \mathbb{F}_t$:
      i. use $[P_t]^{\mathsf{A}}$ and $[N_{t,i}]^{\mathsf{A}}$ to compute $\beta := P_t\big(\alpha, p_0(N_{t,1}(\alpha)), \ldots, p_0(N_{t,\mathsf{c}_{\mathbf{N}}(t)}(\alpha))\big)$
      ii. set $p_1(\alpha) := \beta$
   (b) $B := \textsc{SubspaceInterp}(I_\ell, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, p_1)$
   (c) $d := \deg(P_t(x, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \ldots, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathsf{c}_{\mathbf{N}}(t)})}))$
   (d) $\pi_1 := P_{\text{VRS}}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{H_t}, \mathcal{O}_{H_t}, d, B\big)$
4. Proof that A is consistent with $(\mathtt{x}, 1^t)$:
   (a) compute $A_{\mathtt{x}}$, the low-degree extension of the function $f_{\mathtt{x}} \colon I_{t,\log|\mathtt{x}|} \to \{0,1\}$ defined by $f_{\mathtt{x}}(\alpha_i) := \mathtt{x}_i$ where α_i is the i-th element in $I_{t,\log|\mathtt{x}|}$
   (b) $\pi_{\mathsf{c}} := P_{\text{VRS}}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, A - A_{\mathtt{x}}\big)$
5. $\pi := (p_0, \pi_0, p_1, \pi_1, \pi_{\mathsf{c}})$
6. output π

notes:
- reference: Construction 8.3
- see Lemma C.8 for complexity bounds on queries and randomness
- see Algorithm 9 for V_{sACSP}, the corresponding verifier

Algorithm 3 P_{VRS}

inputs:
- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- $\mathcal{B}_H$ and $\mathcal{O}_H$, basis and offset for the λ-dimensional affine subspace $H \subseteq \mathbb{F}_{2^\ell}$
- d, positive integer indicating what "low-degree" means
- $P \colon \mathbb{F}_{2^\ell} \to \mathbb{F}_{2^\ell}$, polynomial whose evaluation over S is in $\text{VRS}\big(\mathbb{F}_{2^\ell}, S, H, d\big)$

randomness:
- no randomness used

output:
- $\pi = (\widetilde{p}, \widetilde{\pi})$, proof of proximity to $\text{VRS}\big(\mathbb{F}_{2^\ell}, S, H, d\big)$ for the evaluation of P over S

1. $\widetilde{P} := \textsc{SubspaceDivide}(I_\ell, \mathcal{B}_H, \mathcal{O}_H, P)$
2. $\widetilde{p} := \textsc{SubspaceEval}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \widetilde{P})$
3. $\widetilde{\pi} := P_{\text{RS}}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d - 2^\lambda, \widetilde{P}\big)$
4. output $(\widetilde{p}, \widetilde{\pi})$

notes:
- see Lemma C.5 for complexity bounds on the proof length
- see Algorithm 12 for V_{VRS}, the corresponding verifier
- since the evaluation of P over S is in $\text{VRS}\big(\mathbb{F}_{2^\ell}, S, H, d\big)$, it must be that $d \geq |H|$

Algorithm 4 P_{RS}

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- d, positive integer indicating what "low-degree" means
- $P \colon \mathbb{F}_{2^\ell} \to \mathbb{F}_{2^\ell}$, polynomial whose evaluation over S is in $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$

randomness:

- no randomness used

output:

- π, proof of proximity to $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$ for the evaluation of P over S

1. if $\kappa \leq \eta$, then output $\pi :=\perp$
2. if $\kappa > \eta$, then:
   (a) $d_{\kappa,\eta} := |S|/2^\eta - 1 = 2^\kappa/2^\eta - 1$
   (b) if $d < d_{\kappa,\eta}$, then output $\pi := P_{\text{RS},<}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d, P)$
   (c) if $d = d_{\kappa,\eta}$, then output $\pi := P_{\text{RS},=}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, P)$
   (d) if $d > d_{\kappa,\eta}$, then output $\pi := P_{\text{RS},>}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d, P)$

notes:

- see Lemma C.4 for complexity bounds on the proof length
- see Algorithm 13 for V_{RS}, the corresponding verifier
- if $\kappa \leq \eta$, then there is no need to output a proximity proof, because V_{RS} will read the entire implicit input to decide whether the implicit input is in $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$ or not

Algorithm 5 $P_{\mathrm{RS},>}$

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- d, positive integer, greater than $|S|/2^\eta - 1$, indicating what "low-degree" means
- $P\colon \mathbb{F}_{2^\ell} \to \mathbb{F}_{2^\ell}$, polynomial whose evaluation over S is in $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$

randomness:

- no randomness used

output:

- $\pi = \big((p_0, \pi_0), \dots, (p_{2^\eta-1}, \pi_{2^\eta-1})\big)$, proof of proximity to $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$ for the evaluation of P over S

1. if $d \geq |S|$, then output $\pi := \perp$
2. if $d < |S|$, then:
   (a) $d_{\kappa,\eta} := |S|/2^\eta - 1 = 2^\kappa/2^\eta - 1$
   (b) $m_{\kappa,\eta,d} := \lfloor (d+1)/(d_{\kappa,\eta}+1) \rfloor$
   (note that $m_{\kappa,\eta,d} \in \{0, \dots, 2^\eta\}$, because $2^\eta \cdot (d_{\kappa,\eta}+1) = |S|$)
   (c) $P_{\mathsf{rest}}(x) := P(x)$
   (d) for $i = 0, \dots, m_{\kappa,\eta,d} - 1$:
      i. $P_i(x) := P_{\mathsf{rest}}(x) \mod x^{d_{\kappa,\eta}+1}$
      ii. $p_i := \textsc{SubspaceEval}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, P_i)$
      iii. $\pi_i := P_{\mathrm{RS},=}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, P_i)$
      iv. $P_{\mathsf{rest}}(x) := \big(P_{\mathsf{rest}}(x) - P_i(x)\big)/x^{d_{\kappa,\eta}+1}$
   (e) $h := m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1)$
   (f) if $h < d+1$, then:
      i. $P_{m_{\kappa,\eta,d}}(x) := P_{\mathsf{rest}}$
      ii. $p_{m_{\kappa,\eta,d}} := \textsc{SubspaceEval}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, P_{m_{\kappa,\eta,d}})$
      iii. $\pi_{m_{\kappa,\eta,d}} := P_{\mathrm{RS},<}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d-h, P_{m_{\kappa,\eta,d}})$
3. $\pi := \big((p_0, \pi_0), \dots, (p_{2^\eta-1}, \pi_{2^\eta-1})\big)$
   (any p_i or π_i that was not assigned is set to $\perp$)
4. output π

notes:

- see Lemma C.3 for complexity bounds on the proof length
- see Algorithm 14 for $V_{\mathrm{RS},>}$, the corresponding verifier
- if $d \geq |S|$, every function $p\colon S \to \mathbb{F}_{2^\ell}$ is in $\mathrm{RS}(\mathbb{F}_{2^\ell}, S, d)$ (cf. Step 1)

Algorithm 6 $P_{\text{RS},<}$

inputs:
- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- d, positive integer, less than $|S|/2^\eta - 1$, indicating what "low-degree" means
- $P \colon \mathbb{F}_{2^\ell} \to \mathbb{F}_{2^\ell}$, polynomial whose evaluation over S is in $\text{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$

randomness:
- no randomness used

output:
- $\pi = (\pi_1, \pi_2)$, proof of proximity to $\text{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$ for the evaluation of P over S

1. $d_{\kappa,\eta} := |S|/2^\eta - 1 = 2^\kappa/2^\eta - 1$
2. $Q(x) := x^{d_{\kappa,\eta}-d} \in \mathbb{F}_{2^\ell}[x]$
3. $P'(x) := P(x) \cdot Q(x)$
4. $\pi_1 := P_{\text{RS},=}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, P)$
5. $\pi_2 := P_{\text{RS},=}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, P')$
6. $\pi := (\pi_1, \pi_2)$
7. output π

notes:
- see Lemma C.2 for complexity bounds on the proof length
- see Algorithm 15 for $V_{\text{RS},<}$, the corresponding verifier

Algorithm 7 $P_{\mathrm{RS},=}$

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_L = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_L$, basis and offset for the κ-dimensional affine subspace $L \subseteq \mathbb{F}_{2^\ell}$
- $P \colon \mathbb{F}_{2^\ell} \to \mathbb{F}_{2^\ell}$, polynomial whose evaluation over L is in $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big)$

randomness:

- no randomness used

output:

- $\pi = (f, \Pi)$, proof of proximity to $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big)$ for the evaluation of P over S

1. if $\kappa \leq \kappa_0$, then output $\pi := \perp$
2. if $\kappa > \kappa_0$, then:
   (a) $\mathcal{B}_{L_0} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma})$, $\mathcal{O}_{L_0} := \mathcal{O}_L$
   (b) $[Z_{L_0}]^\mathsf{A} := \textsc{FindSubspPoly}(I_\ell, \mathcal{B}_{L_0}, \mathcal{O}_{L_0})$
   (c) $\mathcal{B}_{L_0'} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu})$, $\mathcal{O}_{L_0'} := \mathcal{O}_L$
   (d) $\mathcal{B}_{L_1} := (a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_\kappa)$, $\mathcal{O}_{L_1} := \mathcal{O}_L$
   (e) $\mathcal{B}_{Z_{L_0}(L_1)} := \big([Z_{L_0}]^\mathsf{A}(a_{\lfloor \kappa/2 \rfloor - \gamma + 1}) + [Z_{L_0}]^\mathsf{A}(0_{\mathbb{F}_{2^\ell}}), \ldots, [Z_{L_0}]^\mathsf{A}(a_\kappa) + [Z_{L_0}]^\mathsf{A}(0_{\mathbb{F}_{2^\ell}})\big)$,
   $\mathcal{O}_{Z_{L_0}(L_1)} := [Z_{L_0}]^\mathsf{A}(\mathcal{O}_{L_1})$
   (f) $L_0 := \mathrm{span}(\mathcal{B}_{L_0}) + \mathcal{O}_{L_0}$
   (g) $L_0' := \mathrm{span}(\mathcal{B}_{L_0'}) + \mathcal{O}_{L_0'}$
   (h) $L_1 := \mathrm{span}(\mathcal{B}_{L_1}) + \mathcal{O}_{L_1}$
   (i) for each $\beta \in L_1$:
      i. $\mathcal{O}_{L_\beta} := \mathcal{O}_{L_0'}$
      ii. if $\beta \in L_0'$, then:
         A. $\mathcal{B}_{L_\beta} :=$ "append $a_{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1}$ to the basis $\mathcal{B}_{L_0'}$"
         B. $L_\beta := \mathrm{span}(\mathcal{B}_{L_\beta}) + \mathcal{O}_{L_\beta} = L_0' \cup (L_0' + a_{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1} + \mathcal{O}_{L_0'})$
      else:
         A. $\mathcal{B}_{L_\beta} :=$ "append $\beta + \mathcal{O}_{L_1}$ to the basis $\mathcal{B}_{L_0'}$"
         B. $L_\beta := \mathrm{span}(\mathcal{B}_{L_\beta}) + \mathcal{O}_{L_\beta} = L_0' \cup (L_0' + \beta)$
   (j) $Q := \textsc{FindBivariate}(I_\ell, \mathcal{B}_{L_0}, \mathcal{O}_{L_0}, P)$
   (k) evaluate $Q(x, y) = P_0(x) + P_1(x)y + \cdots + P_e(x)y^e$ on $\cup_{\beta \in L_1} L_\beta \times \{Z_{L_0}(\beta)\}$ to get f:
      i. for $i = 0, \ldots, e$, $s_i := \textsc{SubspaceEval}(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, \mathcal{O}_{Z_{L_0}(L_1)}, P_i)$
      ii. for each $\beta \in L_1$, $\beta' := [Z_{L_0}]^\mathsf{A}(\beta)$ and $f(\cdot, \beta') := \textsc{SubspaceEval}(I_\ell, \mathcal{B}_{L_\beta}, \mathcal{O}_{L_\beta}, P_\beta)$, where $P_\beta(y) := s_0(\beta') + s_1(\beta')y + \cdots + s_e(\beta')y^e$
   (l) for each $\beta \in L_1$:
      i. $\beta' := [Z_{L_0}]^\mathsf{A}(\beta)$
      ii. $F_{\beta'}(x) := Q(x, \beta')$
      iii. $\pi^{\leftrightarrow}_{\beta'} := P_{\mathrm{RS},=}(I_\ell, \mathcal{B}_{L_\beta}, \mathcal{O}_{L_\beta}, F_{\beta'})$
   (m) for each $\alpha \in L_0'$:
      i. $F_\alpha(y) := Q(\alpha, y)$
      ii. $\pi^{\updownarrow}_\alpha := P_{\mathrm{RS},=}(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, \mathcal{O}_{Z_{L_0}(L_1)}, F_\alpha)$
   (n) $\Pi := \{\pi^{\leftrightarrow}_{\beta'} : \beta' \in Z_{L_0}(L_1)\} \cup \{\pi^{\updownarrow}_\alpha : \alpha \in L_0'\}$
   (o) $\pi := (f, \Pi)$
   (p) output π

notes:

- see Theorem 11.3 for how the parameters $(\eta, \kappa_0, \gamma, \mu)$ can be chosen
- see Lemma C.1 for complexity bounds on the proof length
- see Algorithm 16 for $V_{\mathrm{RS},=}$, the corresponding verifier
- in Step 2g, some computation can be saved since $L_0' = L_0 + (\mathrm{span}(a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}) + \mathcal{O}_{L_0})$
- in Step 2(i)ii, since $\beta \in L_1$, it suffices to check if $\beta + \mathcal{O}_{L_1}$ is in $\mathrm{span}(a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}) - \{0\}$
- in Step 2k, we consider all elements $\alpha \in L_\beta$, so that f has domain $\cup_{\beta \in L_1} L_\beta \times \{Z_{L_0}(\beta)\}$; but, f has domain $\cup_{\beta \in L_1} (L_\beta - (L_0 + \beta + \mathcal{O}_L)) \times \{Z_{L_0}(\beta)\}$ in [BSS08, proof of Proposition 6.9], because the values of f in $\cup_{\beta \in L_1} (L_0 + \beta + \mathcal{O}_L) \times \{Z_{L_0}(\beta)\}$ are never queried; nonetheless, we choose to define f there as well, because doing so simplifies a lot of "index arithmetic" when implementing $V_{\mathrm{RS},=}$ (and when decomposing $V_{\mathrm{RS},=}$ into $Q_{\mathrm{RS},=}$ and $D_{\mathrm{RS},=}$); cf. Footnote 28

B.2 PCP Verifier Specifications

Algorithm 8 V_L

inputs:
- $(\mathtt{x}, 1^t)$, allegedly in the language L

randomness:
- $r_1 \cdots r_{\mathsf{rand}_{\mathrm{PCP}}(\mathtt{x},t)}$

oracle:
- π, probabilistically-checkable proof for the satisfiability of $(\mathtt{x}, 1^t)$

1. $b \leftarrow V^{\pi}_{\mathrm{sACSP}}\big((\mathtt{x}, 1^t)\,;r_1 \cdots r_{\mathsf{rand}_{\mathrm{PCP}}(\mathtt{x},t)}\big)$
2. output b

notes:
- see Algorithm 1 for P_L, the corresponding verifier
- V_{sACSP} has the choice of parameters Params_L hardcoded

Algorithm 9 V_{sACSP}

inputs:

- $(\mathtt{x}, 1^t)$, allegedly in sACSP with the given choice of parameters

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

randomness:

- $r_1 \cdots r_{\mathsf{rand}_{\text{ACSP}}(\mathtt{x},t)}$

oracle:

- $\pi = (p_0, \pi_0, p_1, \pi_1, \pi_{\mathsf{c}})$, probabilistically-checkable proof for the claim "$(\mathtt{x}, 1^t) \in$ sACSP"

1. Parameter instantiation:
   (a) $I_t := \textsc{FindIrrPoly}(1^{f(t)})$
   (b) $\mathcal{B}_{\mathbb{F}_t} := \textsc{FieldBasis}(I_t)$
   (c) $(\mathcal{B}_{H_t}, \mathcal{O}_{H_t}) := \textsc{Find}\mathbf{H}(1^t)$
   (d) for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $[N_{t,i}]^{\mathsf{A}} := \textsc{Find}\mathbf{N}(1^t, i)$
   (e) $[P_t]^{\mathsf{A}} := \textsc{Find}\mathbf{P}(1^t)$
   (f) $(\mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}) := \textsc{Find}\mathbf{I}(1^t, 1^{\log|\mathtt{x}|})$
2. Proximity of p_0 to RS:
   (a) $\delta_{\text{RS}} := (8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$, $s'_{\text{RS}} := 0.5$, $R_{\text{RS}} := \mathsf{rand}_{\text{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\text{RS}}, s'_{\text{RS}})$
   (b) $r'_1 \cdots r'_{R_{\text{RS}}} := r_1 \cdots r_{R_{\text{RS}}}$
   (c) $b_{\text{RS}} := V_{\text{aRS}}^{(p_0,\pi_0)}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\text{RS}}, s'_{\text{RS}}\,;\, r'_1 \cdots r'_{R_{\text{RS}}}\big)$
3. Proximity of p_1 to VRS:
   (a) $d := \deg(P_t(x, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \ldots, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathsf{c}_{\mathbf{N}}(t)})}))$
   (b) $\delta_{\text{VRS}} := \frac{1}{8}$, $s'_{\text{VRS}} := 0.5$, $R_{\text{VRS}} := \mathsf{rand}_{\text{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \delta_{\text{VRS}}, s'_{\text{VRS}})$
   (c) $r'_1 \cdots r'_{R_{\text{VRS}}} := r_1 \cdots r_{R_{\text{VRS}}}$
   (d) $b_{\text{VRS}} := V_{\text{aVRS}}^{(p_1,\pi_1)}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{H_t}, \mathcal{O}_{H_t}, d, \delta_{\text{VRS}}, s'_{\text{VRS}}\,;\, r'_1 \cdots r'_{R_{\text{VRS}}}\big)$
4. Consistency of p_0 with p_1:
   (a) $r_1^{(\alpha)} \cdots r_{f(t)}^{(\alpha)} := r_1 \cdots r_{f(t)}$
   (b) for $i = 1, \ldots, \mathsf{c}_{\mathbf{N}}(t)$, $\alpha_i := [N_{t,i}]^{\mathsf{A}}(\alpha)$
   (c) query p_0 at the following points: $\alpha_1, \ldots, \alpha_{\mathsf{c}_{\mathbf{N}}(t)}$
   (d) query p_1 at the point α
   (e) $\omega := [P_t]^{\mathsf{A}}\big(\alpha, p_0(\alpha_1), \ldots, p_0(\alpha_{\mathsf{c}_{\mathbf{N}}(t)})\big)$
   (f) if $(\omega = p_1(\alpha))$, then $b_{\mathsf{consist}} := 1$, else $b_{\mathsf{consist}} := 0$
5. Consistency of p_0 with the instance $(\mathtt{x}, 1^t)$:
   (a) compute $A_{\mathtt{x}}$, the low-degree extension of the function $f_{\mathtt{x}} \colon I_{t,\log|\mathtt{x}|} \to \{0,1\}$ defined by $f_{\mathtt{x}}(\alpha_i) := \mathtt{x}_i$ where α_i is the i-th element in $I_{t,\log|\mathtt{x}|}$
   (b) $\delta_{\mathsf{c}} := (8\sum_{i=1}^{\mathsf{c}_{\mathbf{N}}(t)} \deg(N_{t,i}))^{-1}$, $s'_{\mathsf{c}} := 0.5$, $R_{\mathsf{c}} := \mathsf{rand}_{\text{aVRS}}(f(t), f(t), \log|\mathtt{x}|, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}})$
   (c) $r'_1 \cdots r'_{R_{\mathsf{c}}} := r_1 \cdots r_{R_{\mathsf{c}}}$
   (d) $b_{\mathsf{inst}} := V_{\text{aVRS}}^{(p_0 - p_{\mathtt{x}},\pi_{\mathsf{c}})}\big(I_t, \mathcal{B}_{\mathbb{F}_t}, 0_{\mathbb{F}_t}, \mathcal{B}_{I_{t,\log|\mathtt{x}|}}, \mathcal{O}_{I_{t,\log|\mathtt{x}|}}, 2^{\mathsf{m}_{\mathbf{H}}(t)} - 1, \delta_{\mathsf{c}}, s'_{\mathsf{c}}\,;\, r'_1 \cdots r'_{R_{\mathsf{c}}}\big)$, where $p_{\mathtt{x}}$ is the evaluation of $A_{\mathtt{x}}$ (note that $p_0 - p_{\mathtt{x}}$ can be simulated with access to p_0 and $p_{\mathtt{x}}$, and $p_{\mathtt{x}}$ can be simulated by evaluating $A_{\mathtt{x}}$)
6. $b := b_{\text{RS}} \wedge b_{\text{VRS}} \wedge b_{\mathsf{consist}} \wedge b_{\mathsf{inst}}$
7. output b

notes:

- reference: Construction 8.5
- see Lemma C.8 for complexity bounds on queries and randomness
- see Lemma 12.23 for the "non-adaptive decomposition" of V_{sACSP} into Q_{ACSP} and D_{ACSP}
- see Algorithm 2 for P_{sACSP}, the corresponding prover
- to amplify soundness beyond 1/2, there is of course no need to perform again Step 1, but one does need to re-draw fresh randomness for the other steps (proximity to RS, proximity to VRS, consistency check, and consistency with the instance)

Algorithm 10 V_{aVRS}

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- $\mathcal{B}_H$ and $\mathcal{O}_H$, basis and offset for the λ-dimensional affine subspace $H \subseteq \mathbb{F}_{2^\ell}$
- d, positive integer indicating what "low-degree" means
- δ, proximity parameter
- s', target soundness

randomness:

- $r_1 \cdots r_{\mathsf{rand}_{\text{aVRS}}(\ell,\kappa,\lambda,d,\delta,s')}$

oracles:

- p, function from S to $\mathbb{F}_{2^\ell}$
- $\pi = (\tilde{p}, \tilde{\pi})$, proof of proximity to $\text{VRS}\big(\mathbb{F}_{2^\ell}, S, H, d\big)$ for the function p

1. $m := \left\lceil \frac{\log(1-s')}{\log(1-s_{\text{VRS}}(\delta,2^\kappa))}) \right\rceil$
2. for $i = 1, \ldots, m$:
   (a) $r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)} := r_{(i-1)\cdot\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)+1} \cdots r_{i\cdot\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)}$
   (b) $b_i := V_{\text{VRS}}^{(p,\pi)}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S, \mathcal{B}_H, \mathcal{O}_H, d\,;\, r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\text{VRS}}(\ell,\kappa,\lambda,d)}\big)$
3. $b := b_1 \wedge \cdots \wedge b_m$
4. output b

notes:

- see Lemma C.7 for complexity bounds on queries and randomness
- see Lemma 12.22 for the "non-adaptive decomposition" of V_{aVRS} into Q_{aVRS} and D_{aVRS}
- see Algorithm 3 for P_{VRS}, the corresponding prover (which is the same as that for V_{VRS})
- soundness amplification in Step 2a is done "directly", without attention to randomness efficiency
- from Theorem 11.1, we know that $\left\lceil \frac{\log(1-s')}{\log(1-s_{\text{VRS}}(\delta,2^\kappa))}) \right\rceil \leq \left\lceil \frac{1}{s_{\text{VRS}}(\delta,2^\kappa)} \right\rceil \leq \left\lceil \frac{1}{\delta}\left(2 + \frac{2^{\eta+1}+\kappa^{\log c}}{1-2^{-\eta}}\right) \right\rceil$

Algorithm 11 V_{aRS}

inputs:
- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- d, positive integer indicating what "low-degree" means
- δ, proximity parameter
- s', target soundness

randomness:
- $r_1 \cdots r_{\mathsf{rand}_{\text{aRS}}(\ell,\kappa,d,\delta,s')}$

oracles:
- p, function from S to $\mathbb{F}_{2^\ell}$
- π, proof of proximity to $\text{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$ for the function p

1. $m := \left\lceil \frac{\log(1-s')}{\log(1-s_{\text{RS}}(\delta,2^\kappa)}) \right\rceil$
2. for $i = 1, \ldots, m$:
   (a) $r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\text{RS}}(\ell,\kappa,d)} := r_{(i-1)\cdot\mathsf{rand}_{\text{RS}}(\ell,\kappa,d)+1} \cdots r_{i\cdot\mathsf{rand}_{\text{RS}}(\ell,\kappa,d)}$
   (b) $b_i := V_{\text{RS}}^{(p,\pi)}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d\,;\, r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\text{RS}}(\ell,\kappa,d)}\big)$
3. $b := b_1 \wedge \cdots \wedge b_m$
4. output b

notes:
- see Lemma C.6 for complexity bounds on queries and randomness
- see Lemma 12.21 for the "non-adaptive decomposition" of V_{aRS} into Q_{aRS} and D_{aRS}
- see Algorithm 4 for P_{RS}, the corresponding prover (which is the same as that for V_{RS})
- soundness amplification in Step 2a is done "directly", without attention to randomness efficiency
- from Theorem 11.1, we know that $\left\lceil \frac{\log(1-s')}{\log(1-s_{\text{RS}}(\delta,2^\kappa)}) \right\rceil \leq \left\lceil \frac{1}{s_{\text{RS}}(\delta,2^\kappa)} \right\rceil \leq \left\lceil \frac{1}{\delta}\left(1 + \frac{2^{\eta+1}+\kappa^{\log \mathsf{c}}}{1-2^{-\eta}}\right) \right\rceil$

Algorithm 12 V_{VRS}

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- $\mathcal{B}_H$ and $\mathcal{O}_H$, basis and offset for the λ-dimensional affine subspace $H \subseteq \mathbb{F}_{2^\ell}$
- d, positive integer indicating what "low-degree" means

randomness:

- $r_1 \cdots r_{\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d)}$

oracles:

- p, function from S to $\mathbb{F}_{2^\ell}$
- $\pi = (\widetilde{p}, \widetilde{\pi})$, proof of proximity to $\mathrm{VRS}\big(\mathbb{F}_{2^\ell}, S, H, d\big)$ for the function p

1. Proximity test of $\widetilde{p}$ to $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, S, d - |H|\big)$:
   (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) $b_{\mathrm{RS}} := V_{\mathrm{RS}}^{(\widetilde{p},\widetilde{\pi})}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d - |H|\,;\, r'_1 \cdots r'_{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d-|H|)}\big)$
2. Consistency check between p and $\widetilde{p}$:
   (a) $r_1^{(\alpha)} \cdots r_\kappa^{(\alpha)} := r_1 \cdots r_\kappa$
   (b) $\alpha := \textsc{GetRandElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, (r_1^{(\alpha)}, \ldots, r_\kappa^{(\alpha)}))$
   (c) query α to p
   (d) query α to $\widetilde{p}$
   (e) $[Z_H]^{\mathsf{A}} := \textsc{FindSubspPoly}(I_\ell, \mathcal{B}_H, \mathcal{O}_H)$
   (f) $\omega := [Z_H]^{\mathsf{A}}(\alpha)$
   (g) if $\big(p(\alpha) = \omega \cdot \widetilde{p}(\alpha)\big)$, then $b_{\mathsf{consist}} := 1$, else $b_{\mathsf{consist}} := 0$
3. $b := b_{\mathrm{RS}} \wedge b_{\mathsf{consist}}$
4. output b

notes:

- see Lemma C.5 for complexity bounds on queries and randomness
- see Lemma 12.20 for the "non-adaptive decomposition" of V_{VRS} into Q_{VRS} and D_{VRS}
- see Algorithm 3 for P_{VRS}, the corresponding prover
- it must be that $d \geq |H|$, otherwise p is certainly not in $\mathrm{VRS}\big(\mathbb{F}_{2^\ell}, S, H, d\big)$

Algorithm 13 V_{RS}

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- d, positive integer indicating what "low-degree" means

randomness:

- $r_1 \cdots r_{\mathsf{rand}_{\text{RS}}(\ell,\kappa,d)}$

oracles:

- p, function from S to $\mathbb{F}_{2^\ell}$
- π, proof of proximity to $\text{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$ for the function p

1. if $\kappa \leq \eta$, then:
   (a) for $i = 1, \ldots, 2^\kappa$:
      i. $\alpha_i := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, i)$
      ii. query α_i to p
   (b) $P := \textsc{SubspaceInterp}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, (p(\alpha_1), \ldots, p(\alpha_{2^\kappa})))$
   (c) $\tilde{d} := \deg(P)$
   (d) if $\tilde{d} \leq d$, output 1, else output 0
2. if $\kappa > \eta$, then:
   (a) $d_{\kappa,\eta} := |S|/2^\eta - 1 = 2^\kappa/2^\eta - 1$
   (b) if $d < d_{\kappa,\eta}$, then output $b := V_{\text{RS},<}^{(p,\pi)}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d\,;\, r_1 \cdots r_{\mathsf{rand}_{\text{RS},<}(\ell,\kappa,d)}\big)$
   (c) if $d = d_{\kappa,\eta}$, then output $b := V_{\text{RS},=}^{(p,\pi)}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S\,;\, r_1 \cdots r_{\mathsf{rand}_{\text{RS},=}(\ell,\kappa)}\big)$
   (d) if $d > d_{\kappa,\eta}$, then output $b := V_{\text{RS},>}^{(p,\pi)}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d\,;\, r_1 \cdots r_{\mathsf{rand}_{\text{RS},>}(\ell,\kappa,d)}\big)$

notes:

- see Lemma C.4 for complexity bounds on queries and randomness
- see Lemma 12.19 for the "non-adaptive decomposition" of V_{RS} into Q_{RS} and D_{RS}
- see Algorithm 4 for P_{RS}, the corresponding prover
- if $\kappa \leq \eta$, then V_{RS} can just check directly whether p is in $\text{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$ or not

Algorithm 14 $V_{\mathrm{RS},>}$

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subset $S \subseteq \mathbb{F}_{2^\ell}$
- d, positive integer, greater than $|S|/2^\eta - 1$, indicating what "low-degree" means

randomness:

- $r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},>}(\ell,\kappa,d)}$

oracles:

- p, function from S to $\mathbb{F}_{2^\ell}$
- $\pi = \big((p_0, \pi_0), \ldots, (p_{2^\eta-1}, \pi_{2^\eta-1})\big)$, proof of proximity to $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$ for the function p

1. if $d \geq |S|$, then output $b := 1$
2. if $d < |S|$, then:
   (a) $d_{\kappa,\eta} := |S|/2^\eta - 1 = 2^\kappa/2^\eta - 1$
   (b) $m_{\kappa,\eta,d} := \lfloor (d+1)/(d_{\kappa,\eta}+1) \rfloor$
   (c) $r_1^{(\tau)} \cdots r_\kappa^{(\tau)} := r_1 \cdots r_\kappa$
   (d) $\tau := \textsc{GetRandElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, (r_1^{(\tau)}, \ldots, r_\kappa^{(\tau)}))$
   (e) query τ to p
   (f) for $i = 0, \ldots, m_{\kappa,\eta,d} - 1$:
      i. $r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$
      ii. query τ to p_i
      iii. $b_i := V_{\mathrm{RS},=}^{(p_i,\pi_i)}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d_{\kappa,\eta}\,;\, r_1^{(i)} \cdots r^{(i)}_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}\big)$
   (g) if $m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1) = d+1$, then:
      i. $\omega_\tau := \sum_{i=0}^{m-1} \tau^{i(d_{\kappa,\eta}+1)} \cdot p_i(\tau)$
      ii. if $p(\tau) = \omega_\tau$, then $b_{\mathsf{consist}} := 1$, else $b_{\mathsf{consist}} := 0$
      iii. $b := b_0 \wedge \cdots \wedge b_{m_{\kappa,\eta,d}-1} \wedge b_{\mathsf{consist}}$
      iv. output b
   (h) if $m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1) < d+1$, then:
      i. $d_{\kappa,\eta,d} := d - m_{\kappa,\eta,d} \cdot (d_{\kappa,\eta}+1)$
      ii. $r_1^{(m_{\kappa,\eta,d})} \cdots r^{(m_{\kappa,\eta,d})}_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})} := r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})}$
      iii. $b_{m_{\kappa,\eta,d}} := V_{\mathrm{RS},<}^{(p_{m_{\kappa,\eta,d}},\pi_{m_{\kappa,\eta,d}})}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S, d_{\kappa,\eta,d}\,;\, r_1^{(m_{\kappa,\eta,d})} \cdots r^{(m_{\kappa,\eta,d})}_{\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d_{\kappa,\eta,d})}\big)$
      iv. query τ to $p_{m_{\kappa,\eta,d}}$
      v. $\omega_\tau := \sum_{i=0}^{m_{\kappa,\eta,d}} \tau^{i(d_{\kappa,\eta}+1)} \cdot p_i(\tau)$
      vi. if $p(\tau) = \omega_\tau$, then $b_{\mathsf{consist}} := 1$, else $b_{\mathsf{consist}} := 0$
      vii. $b := b_0 \wedge \cdots \wedge b_{m_{\kappa,\eta,d}-1} \wedge b_{m_{\kappa,\eta,d}} \wedge b_{\mathsf{consist}}$
      viii. output b

notes:

- see Lemma C.3 for complexity bounds on queries and randomness
- see Lemma 12.18 for the "non-adaptive decomposition" of $V_{\mathrm{RS},>}$ into $Q_{\mathrm{RS},>}$ and $D_{\mathrm{RS},>}$
- see Algorithm 5 for $P_{\mathrm{RS},>}$, the corresponding prover
- if $d \geq |S|$, every function $p\colon S \to \mathbb{F}_{2^\ell}$ is in $\mathrm{RS}(\mathbb{F}_{2^\ell}, S, d)$ (cf. Step 1)
- if $d < |S|$, then $\lfloor (d+1)/(d_{\kappa,\eta}+1) \rfloor \in \{0, \ldots, 2^\eta\}$, because $|S| = 2^\eta \cdot (d_{\kappa,\eta}+1)$

Algorithm 15 $V_{\text{RS},<}$

inputs:
- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S$ and $\mathcal{O}_S$, basis for the κ-dimensional affine subset $S \subseteq \mathbb{F}_{2^\ell}$
- d, positive integer, less than $|S|/2^\eta - 1$, indicating what "low-degree" means

randomness:
- $r_1 \cdots r_{\mathsf{rand}_{\text{RS},<}(\ell,\kappa,d)}$

oracles:
- p, function from S to $\mathbb{F}_{2^\ell}$
- $\pi = (\pi_1, \pi_2)$, proof of proximity to $\text{RS}\big(\mathbb{F}_{2^\ell}, S, d\big)$ for the function p

1. $d_{\kappa,\eta} := |S|/2^\eta - 1 = 2^\kappa/2^\eta - 1$
2. $r'_1 \cdots r'_{\mathsf{rand}_{\text{RS},=}(\ell,\kappa)} := r_1 \cdots r_{\mathsf{rand}_{\text{RS},=}(\ell,\kappa)}$
3. $b_1 := V_{\text{RS},=}^{(p,\pi_1)}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S\,;\, r'_1 \cdots r'_{\mathsf{rand}_{\text{RS},=}(\ell,\kappa)}\big)$
4. $r''_1 \cdots r''_{\mathsf{rand}_{\text{RS},=}(\ell,\kappa)} := r_1 \cdots r_{\mathsf{rand}_{\text{RS},=}(\ell,\kappa)}$
5. compute the polynomial $Q(x) := x^{d_{\kappa,\eta}-d} \in \mathbb{F}_{2^\ell}[x]$
6. "$p' := p \cdot Q$" (see notes below)
7. $b_2 := V_{\text{RS},=}^{(p',\pi_2)}\big(I_\ell, \mathcal{B}_S, \mathcal{O}_S\,;\, r''_1 \cdots r''_{\mathsf{rand}_{\text{RS},=}(\ell,\kappa)}\big)$
8. $b := b_1 \wedge b_2$
9. output b

notes:
- see Lemma C.2 for complexity bounds on queries and randomness
- see Lemma 12.17 for the "non-adaptive decomposition" of $V_{\text{RS},<}$ into $Q_{\text{RS},<}$ and $D_{\text{RS},<}$
- see Algorithm 6 for $P_{\text{RS},<}$, the corresponding prover
- in Step 7, we must simulate the implicit input p' (which is also a function from S to $\mathbb{F}_{2^\ell}$), and we do so as follows: for each query α, query α to the (real) implicit input p and then return $p(\alpha) \cdot Q(\alpha)$; note that $Q(\alpha)$ can be evaluated in time that is $\mathsf{poly}(\ell, \log(d_{\kappa,\eta} - d))$

Algorithm 16 $V_{\mathrm{RS},=}$

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_L = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_L$, basis and offset for the κ-dimensional affine subset $L \subseteq \mathbb{F}_{2^\ell}$

randomness:

- $r_1 \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}$

oracles:

- p, function from L to $\mathbb{F}_{2^\ell}$
- $\pi = (f, \Pi)$, proof of proximity to $\mathrm{RS}\big(\mathbb{F}_{2^\ell}, L, |L|/2^\eta - 1\big)$ for the function p

1. if $\kappa \leq \kappa_0$, then:
   (a) for $i = 1, \ldots, 2^\kappa$:
      i. $\alpha_i := \textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_L, \mathcal{O}_L, i)$
      ii. query α_i to p
   (b) $d_{\kappa,\eta} := |L|/2^\eta - 1 = 2^\kappa/2^\eta - 1$
   (c) $P := \textsc{SubspaceInterp}(I_\ell, \mathcal{B}_L, \mathcal{O}_L, (p(\alpha_1), \ldots, p(\alpha_{2^\kappa})))$
   (d) $\tilde{d} := \deg(P)$
   (e) if $\tilde{d} \leq d_{\kappa,\eta}$, output 1, else output 0
2. if $\kappa > \kappa_0$, then:
   (a) $\mathcal{B}_{L_0} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma})$, $\mathcal{O}_{L_0} := \mathcal{O}_L$
   (b) $[Z_{L_0}]^{\mathsf{A}} := \textsc{FindSubspPoly}(I_\ell, \mathcal{B}_{L_0}, \mathcal{O}_{L_0})$
   (c) $\mathcal{B}_{L_0'} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu})$, $\mathcal{O}_{L_0'} := \mathcal{O}_L$
   (d) if $r_1 = 0$, then do a random row test:
      i. $m := 1 + \lceil \kappa/2 \rceil + \gamma$
      ii. $\mathcal{B}_{L_1} := (a_{\lfloor \kappa/2 \rfloor - \gamma + 1}, \ldots, a_\kappa)$, $\mathcal{O}_{L_1} := \mathcal{O}_L$
      iii. $\beta := \textsc{GetRandElt}(I_\ell, \mathcal{B}_{L_1}, \mathcal{O}_{L_1}, (r_2, \ldots, r_{1+\lceil \kappa/2 \rceil + \gamma}))$
      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 $\mathcal{B}_{L_\beta} := (a_1, \ldots, a_{\lfloor \kappa/2 \rfloor - \gamma + \mu}, \beta + \mathcal{O}_{L_0'})$
      vi. $\mathcal{O}_{L_\beta} := \mathcal{O}_{L_0'}$
      vii. $\beta' := [Z_{L_0}]^{\mathsf{A}}(\beta)$
      viii. $b := V_{\mathrm{RS},=}^{(\hat{f}|_{\beta'}^{\leftrightarrow}, \pi_{\beta'}^{\leftrightarrow})}\big(I_\ell, \mathcal{B}_{L_\beta}, \mathcal{O}_{L_\beta}\,;\, r_{m+1} \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}\big)$
      ix. output b
   (e) if $r_1 = 1$, then do a random column test:
      i. $m := 1 + \lfloor \kappa/2 \rfloor - \gamma + \mu$
      ii. $\alpha := \textsc{GetRandElt}(I_\ell, \mathcal{B}_{L_0'}, \mathcal{O}_{L_0'}, (r_2, \ldots, r_{1+\lfloor \kappa/2 \rfloor - \gamma + \mu}))$
      iii. $\mathcal{B}_{Z_{L_0}(L_1)} := ([Z_{L_0}]^{\mathsf{A}}(a_{\lfloor \kappa/2 \rfloor - \gamma + 1}) + [Z_{L_0}]^{\mathsf{A}}(0_{\mathbb{F}_{2^\ell}}), \ldots, [Z_{L_0}]^{\mathsf{A}}(a_\kappa) + [Z_{L_0}]^{\mathsf{A}}(0_{\mathbb{F}_{2^\ell}}))$,
      $\mathcal{O}_{Z_{L_0}(L_1)} := [Z_{L_0}]^{\mathsf{A}}(\mathcal{O}_{L_1})$
      iv. $b := V_{\mathrm{RS},=}^{(\hat{f}|_{\alpha}^{\updownarrow}, \pi_{\alpha}^{\updownarrow})}\big(I_\ell, \mathcal{B}_{Z_{L_0}(L_1)}, \mathcal{O}_{Z_{L_0}(L_1)}\,;\, r_{m+1} \cdots r_{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)}\big)$
      v. output b

notes:

- see Theorem 11.3 for how the parameters $(\eta, \kappa_0, \gamma, \mu)$ can be chosen
- see Lemma C.1 for complexity bounds on queries and randomness
- see Lemma 12.16 for the "non-adaptive decomposition" of $V_{\mathrm{RS},=}$ into $Q_{\mathrm{RS},=}$ and $D_{\mathrm{RS},=}$
- see Algorithm 7 for $P_{\mathrm{RS},=}$, the corresponding prover
- in Step 2(d)viii, $\hat{f}|_{\beta'}^{\leftrightarrow}$ is a function with domain $L_\beta \times \{\beta'\}$ that is simulated as follows: for each query $\alpha \in L_\beta$ to $\hat{f}|_{\beta'}^{\leftrightarrow}$, if $Z_{L_0}(\alpha) = \beta'$ then return $p(\alpha)$, else return $f(\alpha, \beta')$
- in Step 2(e)iv, $\hat{f}|_{\alpha}^{\updownarrow}$ is a function with domain $\{\alpha\} \times Z_{L_0}(L_1)$ that is simulated as follows: for each query $\beta' \in Z_{L_0}(L_1)$ to $\hat{f}|_{\alpha}^{\updownarrow}$, if $Z_{L_0}(\alpha) = \beta'$ then return $p(\alpha)$, else return $f(\alpha, \beta')$

B.3 Finite Field Algorithms

In this section we provide the finite-field algorithms that we used in the pseudocode listings in Section B.1 and Section B.2 (and also those in Section 12). These algorithms provide certain basic computations in extension fields of $\mathbb{F}_2$, and almost all are adapted to the important special case of affine subspaces. Throughout, we let S denote a κ-dimensional affine subspace of the appropriate extension field of $\mathbb{F}_2$, specified via a basis $\mathcal{B}_S$ and an offset $\mathcal{O}_S$.

We begin with a summary of the algorithms described in this section, and following are the peudocode listings.

- Set up:
  - $\textsc{FindIrrPoly}(1^\ell)$ outputs an irreducible polynomial I_ℓ of degree ℓ over $\mathbb{F}_2$ with a root x
  - $\textsc{FieldBasis}(I_\ell)$ outputs a (canonical) basis $\mathcal{B}_\mathbb{F}$ for the field $\mathbb{F} := \mathbb{F}_2(\mathsf{x})$
- Selecting field elements of affine subspaces:
  - $\textsc{GetRandElt}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, (r_1, \dots, r_\kappa))$ outputs a field element in the affine subspace S according to the random binary string $(r_1, \dots, r_\kappa)$
  - $\textsc{GetEltWithIndex}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, i)$ outputs the i-th field element in the affine subspace S according to a canonical order
- Multi-point evaluation and interpolation over affine subspaces:
  - $\textsc{SubspaceEval}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, P)$ outputs the evaluation of $P(x)$ over the affine subspace S
  - $\textsc{SubspaceInterp}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, (\alpha_0, \dots, \alpha_{2^\kappa-1}))$ outputs the interpolation of $(\alpha_0, \dots, \alpha_{2^\kappa-1})$ over the affine subspace S
- Some operations involving vanishing polynomials for affine subspaces:
  - $\textsc{FindSubspPoly}(I_\ell, \mathcal{B}_S, \mathcal{O}_S)$ outputs an arithmetic circuit $[Z_S]^{\mathsf{A}}$ for computing the vanishing polynomial $Z_S(x)$ of the affine subspace S
  - $\textsc{FindBivariate}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, P)$ outputs $Q(x, y) := P(x) \bmod (y - Z_S(x))$
  - $\textsc{SubspaceDivide}(I_\ell, \mathcal{B}_S, \mathcal{O}_S, P)$ outputs $\widetilde{P}(x) := P(x)/Z_S(x)$, if $P(x) \equiv 0 \bmod Z_S(x)$

Algorithm 17 FINDIRRPOLY

inputs:
- 1^ℓ, desired degree presented in unary

output:
- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$

1. the algorithm of [Sho88, Corollary 3.2], which runs in time $\mathsf{poly}(\ell)$

notes:
- the algorithm for finding irreducible polynomials is the only one that we do not spell out explicitly, because we do not plan to use it in practice; indeed, in practice, pre-computed tables of irreducible polynomials work much better

Algorithm 18 FIELDBASIS

inputs:
- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$

output:
- $\mathcal{B}_{\mathbb{F}_{2^\ell}}$, basis for $\mathbb{F}_{2^\ell}$ when viewed as an ℓ-dimensional vector space over $\mathbb{F}_2$

1. output $1, \mathsf{x}, \ldots, \mathsf{x}^{\ell-1}$

notes:
- an optimization that is left to future work is the use of *normal bases* [Gao93]

Algorithm 19 GetRandElt

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S = (a_1, \dots, a_\kappa)$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- $(r_1, \dots, r_\kappa)$, random binary string used to select the element of S

output:

- α, selected element of S

1. $\alpha := \left(\sum_{j=1}^{\kappa} r_j a_j\right) + \mathcal{O}_S$
2. output α

notes:

- the random bits are simply the coefficients of the linear combination

Algorithm 20 GetEltWithIndex

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S = (a_1, \dots, a_\kappa)$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- i, index in $\{1, \dots, 2^\kappa\}$ of the desired element of S

output:

- α, selected element of S

1. $\alpha := (\sum_{j=1}^{\kappa} i_j a_j) + \mathcal{O}_S$
2. output α

notes:

- $i_1 \cdots i_\kappa$ is $(i-1)$ in binary

Algorithm 21 FINDSUBSPPOLY

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$

output:

- $[Z_S]^{\mathsf{A}}$, an $O(\kappa)$ arithmetic circuit for computing $Z_S(x) = \sum_{i=0}^{\kappa} \beta_i x^{2^i} + \gamma$, the vanishing polynomial for the affine subspace S

1. let $[Z_{S_1}]^{\mathsf{A}}$ be the arithmetic circuit for $Z_{S_1}(x) := x \cdot (x + a_1)$
2. if $\kappa = 1$, then:
   (a) use $[Z_{S_1}]^{\mathsf{A}}$ to compute the arithmetic circuit $[Z_S]^{\mathsf{A}}$ for $Z_S(x) := Z_{S_1}(x) + Z_{S_1}(\mathcal{O}_S)$
   (b) output $[Z_S]^{\mathsf{A}}$
3. if $\kappa > 1$, then:
   (a) for $j = 2, \ldots, \kappa$:
      i. use $[Z_{S_{j-1}}]^{\mathsf{A}}$ to compute $\alpha_j := Z_{S_{j-1}}(a_j)$
      ii. use $[Z_{S_{j-1}}]^{\mathsf{A}}$ to compute the arithmetic circuit $[Z_{S_j}]^{\mathsf{A}}$ for $Z_{S_j}(x) := Z_{S_{j-1}}(x^2) + \alpha_j Z_{S_{j-1}}(x)$
   (b) use $[Z_{S_\kappa}]^{\mathsf{A}}$ to compute the arithmetic circuit $[Z_S]^{\mathsf{A}}$ for $Z_S(x) := Z_{S_\kappa}(x) + Z_{S_\kappa}(\mathcal{O}_S)$
   (c) output $[Z_S]^{\mathsf{A}}$

notes:

- the algorithm uses the simple recursion

$$\begin{aligned}
Z_{\mathrm{span}(a_1,\ldots,a_\kappa)}(x) =& Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(x) Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(x + a_\kappa) \\
=& Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(x) \cdot \left(Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(x) + Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(a_\kappa) \right) \\
=& Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(x)^2 + Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(a_\kappa) \cdot Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(x) \\
=& Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(x^2) + Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(a_\kappa) \cdot Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})}(x)
\end{aligned}$$

  together with the fact that

$$Z_{\mathrm{span}(a_1,\ldots,a_\kappa)+\mathcal{O}}(x) = Z_{\mathrm{span}(a_1,\ldots,a_\kappa)}(x + \mathcal{O}) = Z_{\mathrm{span}(a_1,\ldots,a_\kappa)}(x) + Z_{\mathrm{span}(a_1,\ldots,a_\kappa)}(\mathcal{O})$$

Algorithm 22 SUBSPACEEVAL

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- $P(x) = \sum_{i=0}^{d} \beta_i x^i$, polynomial over $\mathbb{F}_{2^\ell}[x]$ with $d < 2^\kappa$

output:

- $(\alpha_0, \ldots, \alpha_{2^\kappa - 1})$, elements in $\mathbb{F}_{2^\ell}$ such that $P(\sum_{i=1}^{\kappa} b_i a_i) = \alpha_{\sum_{i=i}^{\kappa} b_i 2^{i-1}}$ for $b_1 \cdots b_\kappa \in \{0,1\}^\kappa$

1. output $(\alpha_0, \ldots, \alpha_{2^\kappa-1}) := A(I_\ell, \mathcal{B}_S, \mathcal{O}_S, P, 0)$, where A is defined below

Define $A(I_\ell, (a_1, \ldots, a_\kappa), \mathcal{O}, P, j)$ as follows:

1. if $\kappa = 1$, return $P(a_{j2^\kappa})$ and $P(a_{j2^\kappa+1})$
2. if $\kappa > 1$:
   (a) use FINDSUBSPPOLY to find (an arithmetic circuit for) $Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})+\mathcal{O}}(x)$.
   (b) compute $\beta := Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})+\mathcal{O}}(a_{j2^\kappa})$
   (c) divide $P(x)$ by $Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})+\mathcal{O}}(x) - \beta$ to obtain a quotient $Q(x)$ and remainder $R(x)$
   (d) compute $P_0(x) := R(x)$
   (e) compute $P_1(x) := R(x) + Z_{\mathrm{span}(a_1,\ldots,a_{\kappa-1})+\mathcal{O}}(a_\kappa) \cdot Q(x)$
   (f) compute $(\alpha_{j2^\kappa}, \ldots, \alpha_{j2^\kappa+2^{\kappa-1}-1}) := A(I_\ell, (a_1, \ldots, a_{\kappa-1}), \mathcal{O}, P_0, j)$
   (g) compute $(\alpha_{j2^\kappa+2^{\kappa-1}}, \ldots, \alpha_{(j+1)2^\kappa-1}) := A(I_\ell, (a_1, \ldots, a_{\kappa-1}), \mathcal{O}, P_1, j + 2^{\kappa-1})$
   (h) output $(\alpha_{j2^\kappa}, \ldots, \alpha_{(j+1)2^\kappa-1})$

notes:

- see [Mat08, Figure 3.1] for a discussion (of the linear subspace case) and more references
- in an implementation, the vanishing polynomials would be pre-computed first

Algorithm 23 SUBSPACEINTERP

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_S$, basis and offset for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- $(\alpha_0, \ldots, \alpha_{2^\kappa - 1})$, elements in $\mathbb{F}_{2^\ell}$

output:

- $P(x) = \sum_{i=0}^{d} \beta_i x^i$, with $d < 2^\kappa$, such that $P(\sum_{i=1}^{\kappa} b_i a_i) = \alpha_{\sum_{i=i}^{\kappa} b_i 2^{i-1}}$ for $b_1 \cdots b_\kappa \in \{0,1\}^\kappa$

1. output $P := A(I_\ell, \mathcal{B}_S, \mathcal{O}_S, (\alpha_0, \ldots, \alpha_{2^\kappa - 1}), 0)$, where A is defined below

Define $A(I_\ell, (a_1, \ldots, a_\kappa), \mathcal{O}, (\alpha_0, \ldots, \alpha_{2^\kappa - 1}), j)$ as follows:

1. if $\kappa = 1$, return $(x - \alpha_{j2^\kappa}) \cdot (x - \alpha_{j2^\kappa + 1})$
2. if $\kappa > 1$:
   (a) compute $P_0(x) := A(I_\ell, (a_1, \ldots, a_{\kappa-1}), \mathcal{O}, (\alpha_{j2^\kappa}, \ldots, \alpha_{j2^\kappa + 2^{\kappa-1} - 1}), j)$
   (b) compute $P_1(x) := A(I_\ell, (a_1, \ldots, a_{\kappa-1}), \mathcal{O}, (\alpha_{j2^\kappa + 2^{\kappa-1}}, \ldots, \alpha_{(j+1)2^\kappa - 1}), j + 2^{\kappa-1})$
   (c) use FINDSUBSPPOLY to find $Z_{\mathrm{span}(a_1, \ldots, a_{\kappa-1}) + \mathcal{O}}(x)$
   (d) compute $\beta := Z_{\mathrm{span}(a_1, \ldots, a_{\kappa-1}) + \mathcal{O}}(a_{j2^\kappa})$
   (e) compute $R(x) := P_0(x)$
   (f) compute $Q(x) := (P_1(x) - P_0(x))/\beta$
   (g) compute $P(x) := Q(x) \cdot (Z_{\mathrm{span}(a_1, \ldots, a_{\kappa-1}) + \mathcal{O}}(x) - \beta) + R(x)$
   (h) output $P(x)$

notes:

- this is simply the "dual" of Algorithm 22

Algorithm 24 SubspaceDivide

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_S$, basis for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- $P(x) = \sum_{i=0}^{d} \beta_i x^i$, polynomial over $\mathbb{F}_{2^\ell}[x]$ with $d < 2^\ell$ such that $P(x) \equiv 0 \bmod Z_S(x)$

output:

- $\widetilde{P}(x)$, polynomial equal to $P(x)/Z_S(x)$

1. $\widetilde{P}(x) := 0$
2. if $\deg(P) < 2^\kappa$, output $\widetilde{P}(x) := 0$; otherwise continue
3. $Z_S(x) = \sum_{i=0}^{\kappa} \beta_i x^{2^i} + \omega_0 := \textsc{FindSubspPoly}(I_\ell, \mathcal{B}_S, \mathcal{O}_S)$
4. let t be such that $2^t \leq \deg(P) < 2^{t+1}$; note that $\kappa \leq t$
5. $R_0(x) := Z_S(x) - x^{2^\kappa} = \sum_{i=0}^{\kappa-1} \beta_i x^{2^i} + \omega_0$
6. for $j = 1, \ldots, t-k$: compute $R_j(x) := R_{j-1}(x)^2 = R_j(x^2)$
7. $P_0(x) := P(x)$
8. for $j = 1, \ldots, t-\kappa+1$:
   (a) by induction, $\deg(P_{j-1}) < 2^{t-(j-1)+1}$
   (b) if $\deg(P_{j-1}) < 2^{t-(j-1)}$, set $P_j(x) := P_{j-1}(x)$ and go to next iteration of the loop
   (c) if $\deg(P_{j-1}) \geq 2^{t-(j-1)}$, continue this iteration
   (d) write $P_{j-1}(x) = x^{2^{t-j+1}} P^{(1)}_{j-1}(x) + P^{(0)}_{j-1}(x)$ with $\deg(P^{(1)}_{j-1}) \leq \deg(P_{j-1}) - 2^{t-j+1}$ and $\deg(P^{(0)}_{j-1}) < 2^{t-j+1}$ and note that
   $$P_{j-1}(x) = (x^{2^{t-j+1}} + R_{t-\kappa+1-j}(x))P^{(1)}_{j-1}(x) + P'_{j-1}(x) = Z_S(x)^{2^{t-\kappa+1-j}} P^{(1)}_{j-1}(x) + P'_{j-1}(x)$$
   where $P'_{j-1}(x) := R_{t-\kappa+1-j}(x) \cdot P^{(1)}_{j-1}(x) + P^{(0)}_{j-1}(x)$
   (e) set $\widetilde{P}(x) := \widetilde{P}(x) + Z_S(x)^{2^{t-\kappa+1-j}-1} P^{(1)}_{j-1}(x)$
   (f) write $P'_{j-1}(x) = x^{2^{t-j+1}} P^{(1')}_{j-1}(x) + P^{(0')}_{j-1}(x) + P^{(0)}_{j-1}(x)$ with with $\deg_x(P^{(1')}_{j-1}) \leq \deg_x(P_{j-1}) - 2^{t-j+2} + 2^{t-\kappa+1-j}$ and $\deg(P^{(0')}_{j-1}) < 2^{t-j+1}$ and note that
   $$P'_{j-1}(x) = (x^{2^{t-j+1}} + R_{t-\kappa+1-j}(x))P^{(1')}_{j-1}(x) + P_j(x) = Z_S(x)^{2^{t-\kappa+1-j}} P^{(1')}_{j-1}(x) + P_j(x)$$
   where $P_j(x) := R_{t-\kappa+1-j}(x) \cdot P^{(1')}_{j-1}(x) + P^{(0')}_{j-1}(x) + P^{(0)}_{j-1}(x)$ and note that $\deg(P_j) < 2^{t-j+1}$
   (g) set $\widetilde{P}(x) := \widetilde{P}(x) + Z_S(x)^{2^{t-\kappa+1-j}-1} P^{(1')}_{j-1}(x)$
9. output $\widetilde{P}(x)$

notes:

- Step 5 and Step 6 construct a $(t-\kappa+1)$-line table, and in each line of the table there is a large power of 2 of $Z_S(x) - x^{2^\kappa}$
- in each iteration of the loop in Step 8 we divide out a large power of $Z_S(x)$ twice, and these two divisions will jointly halve the degree of the remaining polynomial, and that is why we only need few iterations (i.e., $t-\kappa+1$ iterations)
- in order to compute $Z_S(x)^{2^{t-\kappa+1-j}-1} P^{(1)}_{j-1}(x)$ and $Z_S(x)^{2^{t-\kappa+1-j}-1} P^{(1')}_{j-1}(x)$, respectively in Step 8e and in Step 8g, we should remember that $Z_S(x)^{2^{t-\kappa+1-j}-1} = \prod_{i=0}^{t-\kappa-j} Z_S(x)^{2^i} = \prod_{i=0}^{t-\kappa-j} (x^{2^{t-j}} + R_{t-\kappa-j}(x))$, so that we should evaluate the product "from right to left" (i.e., multiply each of the terms of the product with $P^{(1)}_{j-1}(x)$ or $P^{(1')}_{j-1}(x)$, instead of first evaluating the big product and then multiplying the result with $P^{(1)}_{j-1}(x)$ or $P^{(1')}_{j-1}(x)$)

Algorithm 25 FindBivariate

inputs:

- I_ℓ, irreducible polynomial of degree ℓ over $\mathbb{F}_2$ with root x, inducing the field extension $\mathbb{F}_{2^\ell} := \mathbb{F}_2(\mathsf{x})$
- $\mathcal{B}_S = (a_1, \ldots, a_\kappa)$ and $\mathcal{O}_S$, basis for the κ-dimensional affine subspace $S \subseteq \mathbb{F}_{2^\ell}$
- $P(x) = \sum_{i=0}^{d} \beta_i x^i$, polynomial over $\mathbb{F}_{2^\ell}[x]$

output:

- $Q(x, y)$, bivariate polynomial equal to $P(x) \bmod (y - Z_S(x))$; note that $\deg_x(Q) < 2^\kappa$ and $\deg_y(Q) = \lfloor \frac{\deg(P)}{2^\kappa} \rfloor$

1. if $\deg(P) < 2^\kappa$, output $Q(x, y) := P(x)$; otherwise continue
2. $Z_S(x) = \sum_{i=0}^{\kappa} \beta_i x^{2^i} + \omega_0 :=$ FindSubspPoly$(I_\ell, \mathcal{B}_S, \mathcal{O}_S)$
3. let t be such that $2^t \le \deg(P) < 2^{t+1}$; note that $\kappa \le t$
4. $R_0(x, y) := y + (Z_S(x) - x^{2^\kappa}) = y + \sum_{i=0}^{\kappa-1} \beta_i x^{2^i} + \omega_0$
5. For $j = 1, \ldots, t - k$:
$$R_j(x, y) := \sum_{i=1}^{j} \alpha_{i-1}^2 y^{2^i} + \gamma_{\kappa-1}^2 y + \sum_{i=1}^{\kappa-1} (\gamma_{\kappa-1}^2 \beta_i + \gamma_{i-1}^2) x^{2^i} + \gamma_{\kappa-1}^2 \beta_0 x + \omega_{j-1}^2 + \gamma_{\kappa-1}^2 \omega_0$$
where $R_{j-1}(x, y) = \sum_{i=0}^{j-1} \alpha_i y^{2^i} + \sum_{i=0}^{\kappa-1} \gamma_i x^{2^i} + \omega_{j-1}$
6. $P_0(x, y) := P(x)$
7. For $j = 1, \ldots, t - \kappa + 1$:
   (a) by induction, $\deg_x(P_{j-1}) < 2^{t-(j-1)+1}$
   (b) if $\deg_x(P_{j-1}) < 2^{t-(j-1)}$, set $P_j(x, y) := P_{j-1}(x, y)$ and go to next iteration of the loop
   (c) if $\deg_x(P_{j-1}) \ge 2^{t-(j-1)}$, continue this iteration
   (d) write $P_{j-1}(x, y) = x^{2^{t-j+1}} P_{j-1}^{(1)}(x, y) + P_{j-1}^{(0)}(x, y)$ with $\deg_x(P_{j-1}^{(1)}) \le \deg_x(P_{j-1}) - 2^{t-j+1}$ and $\deg_x(P_{j-1}^{(0)}) < 2^{t-j+1}$
   (e) replace $x^{2^{t-j+1}}$ with $R_{t-\kappa+1-j}(x, y)$ in $P_{j-1}(x, y)$ to obtain $P'_{j-1}(x, y) := P_{j-1}^{(1')}(x, y) + P_{j-1}^{(0)}(x, y)$ with $\deg_x(P_{j-1}^{(1')}) \le \deg_x(P_{j-1}) - 2^{t-j+1} + 2^{\kappa-1}$
   (f) write $P'_{j-1}(x, y) = x^{2^{t-j+1}} P_{j-1}^{(1'')}(x, y) + P_{j-1}^{(0'')}(x, y) + P_{j-1}^{(0)}(x, y)$ with $\deg_x(P_{j-1}^{(1'')}) \le \deg_x(P_{j-1}) - 2^{t-j+2} + 2^{\kappa-1}$ and $\deg_x(P_{j-1}^{(0'')}) < 2^{t-j+1}$
   (g) replace $x^{2^{t-j+1}}$ with $R_{t-\kappa+1-j}(x, y)$ in $P'_{j-1}(x, y)$ to obtain $P_j(x, y)$ with $\deg_x(P_j) < 2^{t-j+1}$
8. output $Q(x, y) := P_{t-\kappa+1}(x, y)$

notes:

- Step 4 and Step 5 construct a $(t - \kappa + 1)$-line table, and in each line of the table there is a polynomial with at most $(t - \kappa + 1)$ y-monomials and κ x-monomials for substitution of a larger power of x (specifically, the i-th line is for substituting $x^{2^{\kappa+i}}$)
- in each iteration of the loop in Step 7 we substitute a large power of x twice, and these two substitutions will jointly halve the degree of the polynomial, and that is why we only need few iterations (i.e., $t - \kappa + 1$ iterations)

C Complexity Analysis

We carefully analyze the query complexity, randomness complexity, and proof length complexity of $(P_{\text{sACSP}}, V_{\text{sACSP}})$, which is the PCP system for sACSP discussed in Section 8;[34] the resulting expressions for the three complexity measures can be easily evaluated numerically for any given set of parameters. Recall that pseudocode listings for the PCP system $(P_{\text{sACSP}}, V_{\text{sACSP}})$ are given in Appendix B.

In this section, we proceed "bottom-up", analyzing each module in Figure 1, one prover-verifier pair at a time. Throughout, we fix a choice of integer parameters $(\eta, \kappa_0, \gamma, \mu)$ satisfying Equation 6 in Theorem 11.3.

C.1 Complexity Analysis of $P_{\text{RS},=}$ and $V_{\text{RS},=}$

The PCPP prover $P_{\text{RS},=}$ is described in Algorithm 7 and the (strong) PCPP verifier $V_{\text{RS},=}$ is described in Algorithm 16.

Lemma C.1 (Complexity Parameters for $P_{\text{RS},=}$ and $V_{\text{RS},=}$)**.** *The following equations hold:*

$$\begin{aligned} \mathsf{query}_{\text{RS},=}(\ell, \kappa) &\leq 2^{\kappa_0} \ , \\ \mathsf{rand}_{\text{RS},=}(\ell, \kappa) &\leq \kappa + (\mu + 2)\log(2\kappa) \ , \\ \mathsf{length}_{\text{RS},=}(\ell, \kappa) &\leq 2^{\kappa}\kappa^{1+\max\{-\gamma+\mu+1,\gamma\}} \ . \end{aligned}$$

Proof. As was already remarked in [BSS08, Propistion 6.8], it is easy to see that the query complexity is as follows:

$$\mathsf{query}_{\text{RS},=}(\ell, \kappa) = \begin{Bmatrix} 2^{\kappa} & \text{if } \kappa \leq \kappa_0 \\ 0 & \text{if } \kappa > \kappa_0 \end{Bmatrix} \leq 2^{\kappa_0} \ .$$

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 C.9 (after appropriately generalizing it to our more general constructions), and our lemma tells us that we can upper bound it follows:

$$\mathsf{rand}_{\text{RS},=}(\ell, \kappa) \leq \begin{Bmatrix} 0 & \text{if } \kappa \leq \kappa_0 \\ \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma+\mu+1, \gamma\}\right) & \text{if } \kappa > \kappa_0 \end{Bmatrix} \leq \kappa + (\mu + 2)\log(2\kappa) \ .$$

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 C.10 (after appropriately generalizing it to our more general constructions), and our lemma tells us that we can upper bound it as follows:

$$\mathsf{length}_{\text{RS},=}(\ell, \kappa) \leq 2^{\kappa}\kappa^{1+\max\{-\gamma+\mu+1,\gamma\}} \ .$$

Of course, if one wishes to numerically evaluate $\mathsf{rand}_{\text{RS},=}(\ell, \kappa)$ or $\mathsf{length}_{\text{RS},=}(\ell, \kappa)$, using the recursive function (in Lemma C.9 or Lemma C.10 respectively) will give the exact value. (Though our bounds are quite tight.) □

[34] 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.

C.2 Complexity Analysis of $P_{\mathrm{RS},<}$ and $V_{\mathrm{RS},<}$

The PCPP prover $P_{\mathrm{RS},<}$ is described in Algorithm 6 and the (strong) PCPP verifier $V_{\mathrm{RS},<}$ is described in Algorithm 15.

Lemma C.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. Immediate from the construction of $P_{\mathrm{RS},<}$ and $V_{\mathrm{RS},<}$. □

C.3 Complexity Analysis of $P_{\mathrm{RS},>}$ and $V_{\mathrm{RS},>}$

The PCPP prover $P_{\mathrm{RS},>}$ is described in Algorithm 5 and the (strong) PCPP verifier $V_{\mathrm{RS},>}$ is described in Algorithm 14. Recall that $d_{\kappa,\eta} := 2^\kappa/2^\eta - 1$.

Lemma C.3 (Complexity Parameters for $P_{\mathrm{RS},>}$ and $V_{\mathrm{RS},>}$)**.** *The following equations hold:*

$$\begin{aligned}\mathsf{query}_{\mathrm{RS},>}(\ell,\kappa,d) &= m_{\kappa,\eta,d}\cdot\mathsf{query}_{\mathrm{RS},=}(\ell,\kappa) + \mathbf{1}_{\kappa,\eta,d}\cdot\mathsf{query}_{\mathrm{RS},<}(\ell,\kappa,d-m_{\kappa,\eta,d}(d_{\kappa,\eta}+1)) + m_{\kappa,\eta,d} + \mathbf{1}_{\kappa,\eta,d} + 1\ ,\\ \mathsf{rand}_{\mathrm{RS},>}(\ell,\kappa,d) &= \max\Big\{\mathsf{rand}_{\mathrm{RS},=}(\ell,\kappa)\,,\ \mathbf{1}_{\kappa,\eta,d}\cdot\mathsf{rand}_{\mathrm{RS},<}(\ell,\kappa,d-m_{\kappa,\eta,d}(d_{\kappa,\eta}+1))\,,\ \kappa\Big\}\ ,\\ \mathsf{length}_{\mathrm{RS},>}(\ell,\kappa,d) &= m_{\kappa,\eta,d}\cdot(2^\kappa+\mathsf{length}_{\mathrm{RS},=}(\ell,\kappa)) + \mathbf{1}_{\kappa,\eta,d}\cdot(2^\kappa+\mathsf{length}_{\mathrm{RS},<}(\ell,\kappa,d-m_{\kappa,\eta,d}(d_{\kappa,\eta}+1)))\ ,\end{aligned}$$

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. Immediate from the construction of $P_{\mathrm{RS},>}$ and $V_{\mathrm{RS},>}$. □

C.4 Complexity Analysis of P_{RS} and V_{RS}

The PCPP prover P_{RS} is described in Algorithm 4 and the (strong) PCPP verifier V_{RS} is described in Algorithm 13.

Lemma C.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. 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}.[35] □

[35] 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!

C.5 Complexity Analysis of P_{VRS} and V_{VRS}

The PCPP prover P_{VRS} is described in Algorithm 3 and the (strong) PCPP verifier V_{VRS} is described in Algorithm 12.

Lemma C.5 (Complexity Parameters for P_{VRS} and V_{VRS})**.** *The following equations hold:*

$$\begin{aligned}\mathsf{query}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d) &= \mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d-2^{\lambda})+2 \ , \\ \mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d) &= \max\left\{\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d-2^{\lambda})\,,\,\kappa\right\} \ , \\ \mathsf{length}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d) &= 2^{\kappa}+\mathsf{length}_{\mathrm{RS}}(\ell,\kappa,d-2^{\lambda}) \ ,\end{aligned}$$

Proof. Immediate from the construction of P_{VRS} and V_{VRS}. □

C.6 Complexity Analysis of V_{aRS} and V_{aVRS}

The ("weak") PCPP verifier V_{aRS} is described in Algorithm 11; its corresponding prover is simply the PCPP prover P_{RS} from Algorithm 4.

Lemma C.6 (Complexity Parameters for V_{aRS})**.** *The following equations hold:*

$$\begin{aligned}\mathsf{query}_{\mathrm{aRS}}(\ell,\kappa,d,\delta,s') &= m_{s_{\mathrm{RS}},\delta,s'}(n)\cdot\mathsf{query}_{\mathrm{RS}}(\ell,\kappa,d) \ , \\ \mathsf{rand}_{\mathrm{aRS}}(\ell,\kappa,d,\delta,s') &= m_{s_{\mathrm{RS}},\delta,s'}(n)\cdot\mathsf{rand}_{\mathrm{RS}}(\ell,\kappa,d) \ .\end{aligned}$$

Proof. Observe that V_{aRS} simply performs naive sequential repetition on V_{RS}. Hence, by Remark 11.2, 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:

$$m_{s_{\mathrm{RS}},\delta,s'}(n)=\left\lceil\frac{\log(1-s')}{s_{\mathrm{RS}}(\delta,n)}\right\rceil \ .$$

where $n=2^{\kappa}$ and s_{RS} is the soundness of V_{RS}. (Recall that Theorem 11.1 gives a lower bound on s_{RS}.)

The query complexity and randomness complexity thus simply increase by the multiplicative factor $m_{s_{\mathrm{RS}},\delta,s'}(n)$. □

The ("weak") PCPP verifier V_{aVRS} is described in Algorithm 10; its corresponding prover is simply the PCPP prover P_{VRS} from Algorithm 3.

Lemma C.7 (Complexity Parameters for V_{aVRS})**.** *The following equations hold:*

$$\begin{aligned}\mathsf{query}_{\mathrm{aVRS}}(\ell,\kappa,\lambda,d,\delta,s') &= m_{s_{\mathrm{VRS}},\delta,s'}(n)\cdot\mathsf{query}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d) \ , \\ \mathsf{rand}_{\mathrm{aVRS}}(\ell,\kappa,\lambda,d,\delta,s') &= m_{s_{\mathrm{VRS}},\delta,s'}(n)\cdot\mathsf{rand}_{\mathrm{VRS}}(\ell,\kappa,\lambda,d) \ .\end{aligned}$$

Proof. Observe that V_{aVRS} simply performs naive sequential repetition on V_{VRS}. Hence, by Remark 11.2, 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:

$$m_{s_{\mathrm{VRS}},\delta,s'}(n)=\left\lceil\frac{\log(1-s')}{s_{\mathrm{VRS}}(\delta,n)}\right\rceil \ .$$

where $n=2^{\kappa}$ and s_{VRS} is the soundness of V_{VRS}. (Recall that Theorem 11.1 gives a lower bound on s_{VRS}.)

The query complexity and randomness complexity thus simply increase by the multiplicative factor $m_{s_{\mathrm{VRS}},\delta,s'}(n)$. □

C.7 Complexity Analysis of P_{sACSP} and V_{sACSP}

The prover P_{sACSP} is described in Algorithm 2 (and also in Construction 8.3) and the verifier V_{sACSP} is described in Algorithm 9 (and also in Construction 8.5).

Lemma C.8 (Complexity Parameters for P_{sACSP} and V_{sACSP}). *The following equations hold:*

$$
\begin{aligned}
\mathsf{query}_{\mathrm{ACSP}}(\mathtt{x},t) = &\,\mathsf{query}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)}-1, \tfrac{1}{8\mathsf{c}_{\mathbf{N}}(t)}, \tfrac{1}{2}) \\
&+ \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \tfrac{1}{8}, \tfrac{1}{2}) \\
&+ (\mathsf{c}_{\mathbf{N}}(t)+1) \\
&+ \mathsf{query}_{\mathrm{aVRS}}(f(t), f(t), \log|\mathtt{x}|, 2^{\mathsf{m}_{\mathbf{H}}(t)}-1, \tfrac{1}{8\mathsf{c}_{\mathbf{N}}(t)}, \tfrac{1}{2}) \ , \\
\mathsf{rand}_{\mathrm{ACSP}}(\mathtt{x},t) = \max\Big\{ &\mathsf{rand}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)}-1, \tfrac{1}{8\mathsf{c}_{\mathbf{N}}(t)}, \tfrac{1}{2}) \, , \\
&\mathsf{rand}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \tfrac{1}{8}, \tfrac{1}{2}) \, , \\
&f(t) \, , \\
&\mathsf{rand}_{\mathrm{aVRS}}(f(t), f(t), \log|\mathtt{x}|, 2^{\mathsf{m}_{\mathbf{H}}(t)}-1, \tfrac{1}{8\mathsf{c}_{\mathbf{N}}(t)}, \tfrac{1}{2}) \Big\} \ , \\
\mathsf{length}_{\mathrm{ACSP}}(\mathtt{x},t) = &\,2^{f(t)} + \mathsf{length}_{\mathrm{aRS}}(f(t), f(t), 2^{\mathsf{m}_{\mathbf{H}}(t)}-1, \tfrac{1}{8\mathsf{c}_{\mathbf{N}}(t)}, \tfrac{1}{2}) \\
&+ 2^{f(t)} + \mathsf{length}_{\mathrm{aVRS}}(f(t), f(t), \mathsf{m}_{\mathbf{H}}(t), d, \tfrac{1}{8}, \tfrac{1}{2}) \\
&+ \mathsf{length}_{\mathrm{aVRS}}(f(t), f(t), \log|\mathtt{x}|, 2^{\mathsf{m}_{\mathbf{H}}(t)}-1, \tfrac{1}{8\mathsf{c}_{\mathbf{N}}(t)}, \tfrac{1}{2}) \ ,
\end{aligned}
$$

where $d := \deg(P_t(x, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,1})}, \ldots, x^{(2^{\mathsf{m}_{\mathbf{H}}(t)}-1)\cdot\deg(N_{t,\mathsf{c}_{\mathbf{N}}(t)})}))$.

Proof. Immediate from the construction of P_{sACSP} and V_{sACSP}. □

C.8 Solving Recursions

We deduce an upper bound for two recursive functions that arise in Appendix C.1: respectively, the upper bounds are on the recursive function giving the amount of randomness used by $V_{\mathrm{RS},=}$ (Lemma C.9) and the recursive function giving the length of the proximity proof generated by $P_{\mathrm{RS},=}$ (Lemma C.10).

Lemma C.9. *The recursion*

$$
r(\kappa) = \begin{cases} 0 & \text{if } \kappa \le \kappa_0 \\ 1 + \max\left\{ \left\lceil \frac{\kappa}{2} \right\rceil + \gamma + r\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 \right) \, , \, \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + r\left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma \right) \right\} & \text{if } \kappa > \kappa_0 \end{cases} \tag{27}
$$

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 6 imply that $\widetilde{r}(k)$ is well-defined for $\kappa > \kappa_0$:

$$
\frac{k}{2} \ge \frac{\kappa_0 + 1}{2} \ge \left\lfloor \frac{\kappa_0 + 1}{2} \right\rfloor \ge \gamma + 1 \ge \gamma + \frac{1}{2} \quad \text{and}
$$

$$\frac{k}{2} \geq \frac{\kappa_0+1}{2} \geq \left\lceil \frac{\kappa_0+1}{2} \right\rceil - \frac{1}{2} \geq -\gamma+\mu+2-\frac{1}{2} = -\gamma+\mu+1+\frac{1}{2} \ ,$$

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 6):

$$\begin{aligned}
&\kappa - \left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma + 1 \right) = \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma - 1 \geq \left\lfloor \frac{\kappa_0+1}{2} \right\rfloor - \gamma - 1 \geq 0 \quad \text{and} \\
&\kappa - \left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 \right) = \left\lceil \frac{\kappa}{2} \right\rceil + \gamma - \mu - 1 \geq 1 \geq 0 \ .
\end{aligned}$$

Finally, we observe that, for every two κ and c with $\kappa \geq 2c$:

$$\begin{aligned}
&\log\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - c \right) \leq \log\left(\frac{\kappa}{2} - c \right) = \log(\kappa - 2c) - 1 \quad \text{and} \\
&\log\left(\left\lceil \frac{\kappa}{2} \right\rceil - c \right) \leq \log\left(\frac{\kappa}{2} - c + 1 \right) = \log(\kappa - 2c + 2) - 1 \ .
\end{aligned}$$

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 27, 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 27, the defining equation for $r(\kappa)$, we get:

$$\begin{aligned}
r(\kappa) &= 1 + \max\left\{ \left\lceil \frac{\kappa}{2} \right\rceil + \gamma + r\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 \right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + r\left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma \right) \right\} \\
&\leq 1 + \max\left\{ \left\lceil \frac{\kappa}{2} \right\rceil + \gamma + \left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 + (\mu+2)\log\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 - 2\max\{-\gamma+\mu+1, \gamma\} \right) \right) \ , \right. \\
&\qquad\qquad \left. \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 0 \right\} \\
&\leq 1 + \max\left\{ \kappa + \mu + 1 + (\mu+2)\log\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 - 2\max\{-\gamma+\mu+1, \gamma\} \right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu \right\} \\
&\leq 1 + \max\left\{ \kappa + \mu + 1 + (\mu+2)\log\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \max\{-\gamma+\mu+1, \gamma\} \right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu \right\} \\
&\leq \max\left\{ \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma+\mu+1, \gamma\} \right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 \right\} \\
&= \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma+\mu+1, \gamma\} \right) \ .
\end{aligned}$$

- *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 27, the defining equation for $r(\kappa)$, we get:

$$r(\kappa) = 1 + \max\left\{ \left\lceil \frac{\kappa}{2} \right\rceil + \gamma + r\left(\left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + 1 \right) \ , \ \left\lfloor \frac{\kappa}{2} \right\rfloor - \gamma + \mu + r\left(\left\lceil \frac{\kappa}{2} \right\rceil + \gamma \right) \right\}$$

$$
\begin{aligned}
&\le 1 + \max\Big\{ \Big\lceil \frac{\kappa}{2} \Big\rceil + \gamma + 0 \ , \\
&\qquad\qquad \Big\lfloor \frac{\kappa}{2} \Big\rfloor - \gamma + \mu + \Big(\Big\lceil \frac{\kappa}{2} \Big\rceil + \gamma + (\mu+2)\log\Big(\Big\lceil \frac{\kappa}{2} \Big\rceil + \gamma - 2\max\{-\gamma+\mu+1,\gamma\}\Big)\Big)\Big\} \\
&= 1 + \max\Big\{ \Big\lceil \frac{\kappa}{2} \Big\rceil + \gamma \, , \, \kappa + \mu + (\mu+2)\log\Big(\Big\lceil \frac{\kappa}{2} \Big\rceil + \gamma - 2\max\{-\gamma+\mu+1,\gamma\}\Big)\Big\} \\
&\le 1 + \max\Big\{ \Big\lceil \frac{\kappa}{2} \Big\rceil + \gamma \, , \, \kappa + \mu + (\mu+2)\log\Big(\Big\lceil \frac{\kappa}{2} \Big\rceil - \max\{-\gamma+\mu+1,\gamma\}\Big)\Big\} \\
&\le \max\Big\{ \Big\lceil \frac{\kappa}{2} \Big\rceil + \gamma + 1 \, , \, \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma+\mu+1,\gamma\} + 2\right) - 1\Big\} \\
&\le \max\Big\{ \Big\lceil \frac{\kappa}{2} \Big\rceil + \gamma + 1 \, , \, \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma+\mu+1,\gamma\}\right)\Big\} \\
&= \kappa + (\mu+2)\log\left(\kappa - 2\max\{-\gamma+\mu+1,\gamma\}\right) \ .
\end{aligned}
$$

- *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 C.10. *The recursive function*

$$
r(\kappa) = \begin{cases} 0 & \text{if } \kappa \le \kappa_0 \\ 2^{\lceil \kappa/2 \rceil + \gamma} \cdot 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1} + 2^{\lceil \kappa/2 \rceil + \gamma} \cdot r\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right) + 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu} \cdot r\left(\lceil \kappa/2 \rceil + \gamma\right) & \text{if } \kappa > \kappa_0 \end{cases} \tag{28}
$$

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 $r(\kappa) \le 2^{\kappa} \cdot \widetilde{s}(\kappa)$. For $\kappa \in \{1, \ldots, \kappa_0\}$, the inequality $r(\kappa) \le 2^{\kappa} \cdot \widetilde{s}(\kappa)$ 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 $\lfloor \kappa/2 \rfloor - \gamma + \mu + 1 \le \kappa_0$ and $\lceil \kappa/2 \rceil + \gamma \le \kappa_0$.* By Equation 28, the defining equation for $r(\kappa)$, we get:

$$
\begin{aligned}
r(\kappa) &= 2^{\mu+1} \cdot 2^{\kappa} + 2^{\lceil \kappa/2 \rceil + \gamma} \cdot r\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right) + 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu} \cdot r\left(\lceil \kappa/2 \rceil + \gamma\right) \\
&\le 2^{\mu+1} \cdot 2^{\kappa} + 2^{\lceil \kappa/2 \rceil + \gamma} \cdot 0 + 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1} \cdot 0 \\
&= 2^{\mu+1} \cdot 2^{\kappa} \\
&\le 2^{\kappa} \cdot \widetilde{s}(\kappa) \ ,
\end{aligned}
$$

  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 $\lceil \kappa/2 \rceil + \gamma \le \kappa_0$.* By Equation 28, the defining equation for $r(\kappa)$, we get:

$$
\begin{aligned}
r(\kappa) &= 2^{\mu+1} \cdot 2^{\kappa} + 2^{\lceil \kappa/2 \rceil + \gamma} \cdot r\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right) + 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu} \cdot r\left(\lceil \kappa/2 \rceil + \gamma\right) \\
&\le 2^{\mu+1} \cdot 2^{\kappa} + 2^{\lceil \kappa/2 \rceil + \gamma} \cdot \Big(2^{\lfloor \kappa/2 \rfloor - \gamma + \mu + 1} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right)\Big) + 2^{\lfloor \kappa/2 \rfloor - \gamma + \mu} \cdot 0
\end{aligned}
$$

$$
\begin{aligned}
&= 2^{\mu+1} \cdot 2^{\kappa} + 2^{\mu+1} \cdot 2^{\kappa} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right) \\
&= 2^{\kappa} \cdot \left(2^{\mu+1} + 2^{\mu+1} \cdot \widetilde{s}\left(\lfloor \kappa/2 \rfloor - \gamma + \mu + 1\right)\right) \\
&\leq 2^{\kappa} \cdot \widetilde{s}(\kappa) \ ,
\end{aligned}
$$

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 28, 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 28, 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 *Proceedings of the 37th International Colloquium on Automata, Languages and Programming*, ICALP '10, pages 152–163, 2010.

[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.

[AV77] Dana Angluin and Leslie G. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. In *Proceedings on 9th Annual ACM Symposium on Theory of Computing*, STOC '77, pages 30–41, 1977.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS '01: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 106–115, Washington, DC, USA, 2001. IEEE Computer Society.

[BC12] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *Proceedings of the 32nd Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '12, 2012.

[BCCT12a] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 326–349, 2012.

[BCCT12b] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. Cryptology ePrint Archive, Report 2011/95, 2012.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *Proceedings of the 10th Theory of Cryptography Conference*, TCC '13, pages ???–???, 2013.

[Ber68] Elwyn R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, New York, 1968.

[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 *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, STOC '91, pages 21–32, 1991.

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 390–420, 1993.

[BG08] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM Journal on Computing*, 38(5):1661–1694, 2008. Preliminary version appeared in CCC '02. We reference the version available online at `http://www.wisdom.weizmann.ac.il/~oded/PS/ua-rev3.ps`.

[BGV12] Zvika Brakerski, Craig Gentry, and Vaikuntanathan Vinod. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 4th Symposium on Innovations in Theoretical Computer Science*, ITCS '12, pages 309–325, 2012.

[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`.

[BOGKW88] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: how to remove intractability assumptions. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, STOC '88, pages 113–131, 1988.

[BP04] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In *Proceedings of the 24th Annual International Cryptology Conference*, CRYPTO '04, pages 273–289, 2004.

[BSCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. In *Proceedings of the 4th Innovations in Theoretical Computer Science Conference*, ITCS '13, 2013.

[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 *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC '04, pages 1–10, 2004.

[BSGH$^+$05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Short PCPs verifiable in polylogarithmic time. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, CCC '05, pages 120–134, 2005.

[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.

[BSW12] Dan Boneh, Gil Segev, and Brent Waters. Targeted malleability: homomorphic encryption for restricted computations. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 350–366, 2012.

[CKV10] Kai-Min Chung, Yael Kalai, and Salil Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Proceedings of the 30th Annual International Cryptology Conference*, CRYPTO '10, pages 483–501, 2010.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 4th Symposium on Innovations in Theoretical Computer Science*, ITCS '12, pages 90–112, 2012.

[CR72] Stephen A. Cook and Robert A. Reckhow. Time-bounded random access machines. In *Proceedings of the 4th Annual ACM Symposium on Theory of Computing*, STOC '72, pages 73–80, 1972.

[CRR11] Ran Canetti, Ben Riva, and Guy N. Rothblum. Two 1-round protocols for delegation of computation. Cryptology ePrint Archive, Report 2011/518, 2011.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Proceedings of the 1st Symposium on Innovations in Computer Science*, ICS '10, pages 310–331, 2010.

[CT12] Alessandro Chiesa and Eran Tromer. Proof-carrying data: Secure computation on untrusted platforms (high-level description). *The Next Wave: The National Security Agency's review of emerging technologies*, 19(2):40–46, 2012.

[CV12] Melissa Chase and Ivan Visconti. Secure database commitments and universal arguments of quasi-knowledge. In *Proceedings of the 32nd Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '12, 2012.

[Dam92] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *Proceedings of the 11th Annual International Cryptology Conference*, CRYPTO '92, pages 445–456, 1992.

[DCL08] Giovanni Di Crescenzo and Helger Lipmaa. Succinct NP proofs from an extractability assumption. In *Proceedings of the 4th Conference on Computability in Europe*, CiE '08, pages 175–185, 2008.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *Proceedings of the 9th International Conference on Theory of Cryptography*, TCC '12, pages 54–74, 2012.

[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 *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, pages 155–164, 2004.

[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.

[Gao93] Shuhong Gao. *Normal Bases over Finite Fields*. PhD thesis, University of Waterloo, 1993.

[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.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *Proceedings of the 30th Annual International Cryptology Conference*, CRYPTO '10, pages 465–482, 2010.

[GGPR12] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. Cryptology ePrint Archive, Report 2012/215, 2012.

[GH11a] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS' 11, pages 107–109, 2011.

[GH11b] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '11, pages 129–148, 2011.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for Muggles. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, STOC '08, pages 113–122, 2008.

[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.

[GSS12a] Craig Gentry, Halevi Shai, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In *Proceedings of the 15th International Conference on Practice and Theory in Public Key Cryptography*, PKC '12, pages 1–16, 2012.

[GSS12b] Craig Gentry, Halevi Shai, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Proceedings of the 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '12, pages 465–482, 2012.

[GVW02] Oded Goldreich, Salil Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 11(1/2):1–53, 2002.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, STOC '11, pages 99–108, 2011.

[HB98] W. C. Huffman and Richard A. Brualdi. *Handbook of Coding Theory*. Elsevier Science Inc., New York, NY, USA, 1998.

[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.

[HT98] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In *Proceedings of the 18th Annual International Cryptology Conference*, CRYPTO '98, pages 408–423, 1998.

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *Proceedings of the Twenty-Second Annual IEEE Conference on Computational Complexity*, CCC '07, pages 278–291, 2007.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, STOC '92, pages 723–732, 1992.

[KR09] Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In *CRYPTO '09: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, pages 143–159, London, UK, 2009. Springer-Verlag.

[KRR12] Yael Kalai, Ran Raz, and Ron Rothblum. Where delegation meets Einstein. Isaac Newton Institute for Mathematical Sciences, Formal and Computational Cryptographic Proofs, 2012.

[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.

[Lip10] Richard J. Lipton. Galactic algorithms. `http://rjlipton.wordpress.com/2010/10/23/galactic-algorithms/`, 2010.

[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, 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 *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science*, MFCS '98, pages 94–116, 1998.

[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.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *Proceedings of the 23rd Annual International Cryptology Conference*, CRYPTO '03, pages 96–109, 2003.

[PR05] Rafael Pass and Alon Rosen. Concurrent non-malleable commitments. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '05, pages 563–572, 2005.

[PS94] Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC '94, pages 194–203, 1994.

[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 *Proceedings of the 24th IEEE Annual Conference on Computational Complexity*, CCC '09, pages 81–92, 2009.

[SBV+12] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. Cryptology ePrint Archive, Report 2012/622, 2012.

[SBW11] Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Toward practical and unconditional verification of remote computations. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS '13, pages 29–29, 2011.

[Sho88] Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '88, pages 283–290, 1988.

[SMBW12] Srinath Setty, Michael McPherson, Andrew J. Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *Proceedings of the 2012 Network and Distributed System Security Symposium*, NDSS '12, pages ???–???, 2012.

[Spi95] Daniel Spielman. *Computationally Efficient Error-Correcting Codes and Holographic Proofs.* PhD thesis, MIT, Mathamatics Department, May 1995.

[SVP+12] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the 21st USENIX Security Symposium*, Security '12, page ???, 2012.

[Sze05] Mario Szegedy. Probabilistic verification and non-approximability. In *Handbook of Combinatorial Optimization*, pages 83–191. Springer US, 2005.

[TRMP12] Justin Thaler, Mike Roberts, Michael Mitzenmacher, and Hanspeter Pfister. Verifiable computation with massively parallel interactive proofs. *CoRR*, abs/1202.1350, 2012.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Proceedings of the 5th Theory of Cryptography Conference*, TCC '08, pages 1–18, 2008.

[vzGG03] Joachim von zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[VZGP01] Joachim Von Zur Gathen and Daniel Panario. Factoring polynomials over finite fields: a survey. *Journal of Symbolic Computation*, 31(1-2):3–17, Jan 2001.

[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