BOIL: Proof-Carrying Data from Accumulation of Correlated Holographic IOPs

Tohru Kohrita, Maksim Nikolaev, and Javier Silva

=nil; Foundation, Limassol, Cyprus

tohru.kohrita@gmail.com
maksim.n@mailbox.org
javier.silva@nil.foundation

Abstract. In this paper, we present a batching technique for oracles corresponding to codewords of a Reed–Solomon code. This protocol is inspired by the round function of the STIR protocol (CRYPTO 2024). Using this oracle batching protocol, we propose a construction of a practically efficient accumulation scheme, which we call BOIL. Our accumulation scheme can be initiated with an arbitrary correlated holographic IOP, leading to a new class of PCD constructions. The results of this paper were originally given as a presentation at zkSummit12.

Keywords: Split Accumulation · IVC · PCD · IOPP · Proof Systems

1 Introduction

Proof-carrying data (PCD) [Chi10] is a cryptographic primitive that enables the dynamic compilation of a distributed computation, where each message is augmented by a short proof verifying that some local condition is met. An ideal PCD construction achieves this goal with minimal additional computation or communication overhead. A specific instance of PCD is incrementally verifiable computation (IVC) [Val08], a cryptographic primitive that allows one to produce a proof of the correctness of an iterative computation in an incremental fashion.

PCD via Recursive Composition. Early constructions of IVC and PCD relied on recursive composition, where each prover attaches a proof to each outgoing message, attesting that all previous local conditions have been met [BCCT12,BCTV14,COS20]. To achieve this, the constructions included a circuit representation of the verifier algorithm within a recursive statement. As a result, PCD was primarily limited to SNARKs – proof systems where the verifier's description is asymptotically smaller than the overall size of the statement being verified.

PCD via Accumulation. Later works [BGH19,BDFG21,BCMS20,BCL+21] showed that it is often possible to avoid the complete recursive proof verification at each iteration. Instead, the most expensive part of the verification can be accumulated outside the recursive statement, and checked only once at the very end of the distributed computation. Verifying the proof of the correctness of the

accumulation is usually much simpler than a full proof verification. The folding approach can be considered an extreme version of accumulation schemes for constructing IVC [KST22,BC23,EG23]. While a very small recursive overhead distinguishes constructions using folding, the accumulation approach allows using a broader class of NARKs as a basic block of the PCD scheme.

However, accumulation and folding schemes usually require the use of additively homomorphic commitment schemes. Therefore, this entire series of results is not compatible with (S)NARKs relying on code-based polynomial commitments, which are not homomorphic. However, even when using a full in-circuit verifier for recursion [COS20], such SNARKs perform excellently and are widely used in practice [Sta21,Teaa,Teab,KPV22]. The work of [BMNW24a] was the first attempt in trying to close this gap, showing how to build an accumulation scheme by postponing the code proximity test of such non-homomorphic constructions to the end. However, their work has significant shortcomings, discussed below in more detail, which our new construction addresses.

1.1 Our contributions

Oracle Batching Protocol. In this paper, we consider SNARKs that are built using a compilation of a Polynomial-IOP and a proximity proof for a linear code. This approach is widely adopted in the industry. Among the reasons, we can highlight the following.

- No need for a trusted ceremony and trapdoors to generate parameters.
- Such systems do not require public key assumptions.
- The system parameters can be adjusted to alter the efficiency trade-off between the prover and the verifier.
- Protocols for code proximity tests, for example, FRI, keep all arithmetic in the same field for prover and verifier, so no field switching is necessary. This means, in particular, that the recursive composition of proofs does not require the use of cycles of elliptic curves.

However, the polynomial commitment schemes derived from proximity proofs are not an additive like KZG [KZG10] or Pedersen [Ped92], which prevents a direct application of standard accumulation results.

The primary goal of FRI [BBHR18] (or any other proximity test) is to distinguish, by querying a function $f : D \to \mathbb{F}$ at a few locations, whether f coincides with the evaluation of some polynomial of degree less than $d < |D|$ on the domain D, or whether it is far in relative Hamming distance from the evaluation of any low-degree polynomial. The batched version of the protocol [BCI+20,Hab22] allows one to perform a proximity test for several functions $f_1, \ldots, f_n$ at once. However, to build more efficient IVC/PCD schemes, we need a reduction for multiple functions that does not require a full-fledged proximity test. Informally, this allows the incremental aggregation of new functions in batches, during the long-running computation. The problem of finding such a reduction was partially solved in [BMNW24a], but with a limitation on the number of

recursive steps allowed. The question of finding a more general solution remained open.

Our first contribution is a batching oracle protocol heavily inspired by the recent STIR protocol [ACFY24a]. Suppose that we have n functions $f_1, \ldots, f_n : D \to \mathbb{F}$, for some $D \subset \mathbb{F}$. The verifier has oracle access to them, and the prover claims that f_i is δ-close to a low-degree polynomial, for all $i = 1, \ldots, n$. The protocol will produce a function $f_{\mathsf{new}} : D' \to \mathbb{F}$, for some other $D' \subset \mathbb{F}$, and reduce the n claims above to the single claim that f_{new} is δ-close to a low-degree polynomial.

This protocol does not require a proximity test, so it is much more efficient than batched FRI. Thus, it can be a base block for various designs such as a split accumulator or a linear combination scheme for PCS [BDFG21]. We consider the first construction in detail. The second follows implicitly, but the formalization is out of the scope of this paper.

Split Accumulation for IOPP. The new oracle batching protocol opens the doors for more efficient aggregation and recursive composition of proofs. Informally speaking, instead of performing a low-degree proximity test on each iteration of recursion, we can use the oracle batching protocol, thereby deferring this expensive test to the very end of the computation.

In the context of IVC/PCD constructions, an important performance metric is the recursive overhead. In the case of SNARKs based on the low-degree proximity test for Reed–Solomon codes, the recursive statement usually includes many hash invocations. The paper [COS20] formally shows how such SNARKs can be used to construct IVC/PCD. The described approach requires representing the verifier as a circuit, in which the lion's share is occupied by hash operations, even when using SNARK-friendly hash functions such as Poseidon [GKR+21]. The proximity test makes the largest contribution to this size: $O(\lambda \cdot c_\delta \log^2 d)$ hash invocations, where λ is a security parameter, d is the corresponding polynomial degree and c_δ depends on the proximity parameter δ.

The paper [BMNW24a] proposes an alternative construction of the IVC/PCD based on a linearity test and exclusively symmetric assumptions. Their linearity test has a better asymptotic estimate of $O(\lambda \cdot c_\delta \log d)$ hash invocations. However, its use entails the problem of distance decay: with each iteration, the provable distance increases, so a much smaller proximity parameter δ/T must be used, where T is the number of iterations. For practically significant parameters, this negates the advantage since $O(\lambda \cdot c_{\delta/T} \log d)$ can be comparable to or even larger than $O(\lambda \cdot c_\delta \log^2 d)$.

Our technique, which we call BOIL (**B**atching **O**racles for **I**OPP from **L**inearity), allows us to get the best of both worlds – logarithmic complexity and the absence of the distance decay problem. It is a theoretical and practical improvement over previous results.

We aim to show that BOIL can be used with various proof systems. Therefore, for formalization, we use the abstraction of correlated Holographic IOPs [CBBZ23]. Roughly speaking, a δ-correlated Holographic IOP (HIOP) [BGK+23] is a holographic proof system in which the verifier has access to an oracle that

checks whether the requested polynomial is close to the Reed–Solomon code. This model captures many protocols (PLONK [GWC19a], Plonky2 [Teaa], RISC Zero [Teab], Redshift [KPV22]) and gives us the flexibility we need. We show that we can build a split accumulation scheme, given a round-by-round (RBR) knowledge sound δ-correlated HIOP, the Oracle batching protocol, and a RBR sound proximity test. The result of [BCL+21] directly yields the IVC/PCD construction.

1.2 Our techniques

As noted above, running a proximity test for the Reed–Solomon code $\mathsf{RS}[\mathbb{F}, D, d]$ within a circuit is expensive, due to the large amount of hashes. Thus, we want to defer as much to the end of the recursive computation as we can, to minimize the size of the in-circuit verifier in the final IVC/PCD construction.

To achieve this, we want to recursively aggregate all these proximity checks for Reed–Solomon into a single one, and perform a proximity test only once at the end; hence we are interested in an oracle batching technique. This was achieved in [BMNW24a], using a simpler oracle batching technique. However, it imposed some limitations on their parameter choices which hindered the practicality of the scheme.

Oracle batching. The construction in [BMNW24a] achieves the goal above by a technique that can be thought of as a single round of the FRI proximity test [BBHR18]: given functions $f_1, \ldots, f_n : D \to \mathbb{F}$, these are combined into a single function $f_{\mathsf{new}} : D \to \mathbb{F}$ by means of a random linear combination, with randomness supplied by the verifier. Informally, if an honestly computed f_{new} is δ-close to $\mathsf{RS}[\mathbb{F}, D, d]$, then with high probability so are $f_1, \ldots, f_n$. The prover sends an oracle for the purported f_{new}. The verifier ensures consistency between $f_1, \ldots, f_n$ and f_{new} by means of spot checks at random points in D. This process can be iterated upon, and at the end the verifier simply checks proximity of the final aggregated function to $\mathsf{RS}[\mathbb{F}, D, d]$. This can be performed directly or by means of a fully fledged proximity test, like FRI or STIR.

This construction has two significant limitations. One is that it requires to work within the unique decoding radius $(1 - \rho)/2$ of the Reed–Solomon code, where ρ is the code rate, to prevent ambiguous (i.e. multiple) decoding which thwarts the security proof of the accumulator. This leads to parameters that are inefficient in practice: the smaller the decoding radius is, the more queries are necessary to guarantee the same level of soundness in the spot checks phase. The other issue is that the distance guarantee degrades with subsequent iterations of this procedure. In more detail, the protocol only checks that

- f_{new} is δ-close to $\mathsf{RS}[\mathbb{F}, D, d]$ by the final proximity test,
- the honest folding of $f_1, \ldots, f_n$ is δ-close to f_{new} by the spot checks.

Hence, overall we can only guarantee that $f_1, \ldots, f_n$ are 2δ-close to the code. More generally, over k recursion steps, we can only guarantee $k\delta$-closeness to the code. From a theoretical point of view, this means that we can only build bounded-depth accumulation from this technique (although [BMNW24a] shows that this is enough to build PCD). From a practical point of view, this makes

us into even more expensive choices of the code's proximity parameter, namely $(1-\delta)/2k$.

These two problems are solved if we try to use a round from STIR [ACFY24a] instead of a round from FRI as our batching technique. It starts exactly as the previous approach, but includes additional steps at the end. Namely, it adds an out-of domain check and quotienting to disambiguate the decoding of f_{new}.

This is very reminiscent of the technique often used to turn FRI into a polynomial commitment scheme [KPV22]. Informally, this ensures that, with high probability, there is only one valid choice of codeword, even if we are in the list decoding regime. Moreover, it also gets rid of the distance decay issue: we can directly prove that if a (possibly dishonest) f_{new} is δ-close to the code, then $f_1, \ldots, f_n$ are δ-close to the code. This allows us to work with the much better proximity parameter $1-\sqrt{\rho}$ (or $1-\rho$, under the so-called Reed–Solomon decoding conjectures [BCI+20, Conjecture 8.4]).

Split accumulation. In order to obtain a PCD scheme, we follow the overall strategy of [BMNW24a]. In particular, it suffices to show that, for a given NARK, its verifier relation admits a split accumulation scheme.

In our setting, the SNARK verification procedure is naturally divided into two parts:

- the Polynomial-IOP–related checks, and
- the verification of a proximity proof.

Additionally, for the latter, we now have an oracle batching protocol. Hence, a natural idea is to split the verifier relation accordingly.

Although the idea is conceptually straightforward, its formalization is technically involved. The main difficulty arises because the relation for the proximity check is defined over oracles, so we cannot simply decouple it. We must first obtain its plain (non-oracle) version. Substituting oracles with vector commitments introduces another subtlety: the underlying vectors may be partially undefined, since proximity testing doesn't guarantee full codeword membership.

Our starting point for constructing a NARK is the framework of δ-correlated IOPs introduced in [BGK+23]. Such IOPs are equipped with an oracle that checks for δ-correlated agreement. Functions $f_1, \ldots, f_n : D \to \mathbb{F}$ are said to be in δ-correlated agreement if each agrees with some codeword in $\mathrm{RS}[\mathbb{F}, D, d]$ on at least a $(1-\delta)$ fraction of points in D, and the agreement subset of D is the same for all f_i. This notion captures an IOP that combines a polynomial check with a proximity (or correlated-agreement) check as part of its final verification.

In Chapter 4, we introduce a series of technical transformations, which will later be used to define the split accumulation scheme in Section 5. Below we provide an informal overview:

- Transformation B takes as input a δ-correlated IOP that checks agreement for n functions and converts it into a δ-correlated IOP that checks proximity to an RS code for a single function. Such a transformation can be directly derived from the oracle batching protocol.

– The paper [BGK+23] shows that a δ-correlated IOP for a relation $\mathcal{R}$ can be combined with a standard IOP for correlated agreement to produce a (regular) IOP for $\mathcal{R}$, while preserving soundness. However, we cannot reuse this transformation directly, since we require two separate verifiers: V_1 for the Polynomial-IOP checks, and V_2 for the proximity-proof verification. To achieve this split verifier, we introduce a modified transformation T.
– Once we obtain a regular IOP, we can apply the BCS transformation [BCS16] to derive a NARK in the random oracle model. Since our verifier is split, we require a modified version of the BCS transformation. Moreover, the verifier V_2 must account for potentially undefined entries in its input vector. To handle these cases, we define the transformations BCST and $\mathsf{BCST}^{\perp}$, respectively.

We show that, under reasonable conditions, these transformations also preserve knowledge soundness (Lemmas 6 and 5). Hence, we can define

$$\mathsf{NARK} = \mathsf{BCST}(\mathsf{T}(\mathsf{B}(\Pi))),$$

and its relaxed version

$$\mathsf{NARK}^{\perp} = \mathsf{BCST}^{\perp}(\mathsf{T}(\mathsf{B}(\Pi))),$$

which serve as the basis for constructing the split accumulation scheme, formally described in Chapter 5.

This results in a construction with an accumulator verifier that does not need to run a full proximity test on each accumulation step. Thus, we avoid the issue of running a full proximity test inside of the final PCD's prover circuit. Moreover, because of the STIR-based oracle batching technique, our construction does not inherit the parameter limitations as in [BMNW24a].

1.3 Related works

Comparison with Folding-based constructions. Nova-like folding schemes [KST22,KS22,KS24] allow efficient IVC constructions but are based on the use of R1CS or CCS arithmetization. Our construction allows the use of the Plonkish arithmetization [GW20b,GW20a], which is very expressive and widely used in practice.

Another important difference is that the EC-based IVC/PCD designs use cycles of curves [KST22,BC23,EG23]. This makes formalization somewhat more complicated [KS23,NBS23]. Moreover it requires the use of non-native arithmetic, which significantly increases the recursive overhead. By avoiding cycles of curves, we get a conceptually more straightforward construction.

Finally, IVC constructions are usually neither succinct nor zero knowledge: the size of the state that the prover must maintain is proportional to the size of the circuit. This significantly complicates the parallelization of proof generation for a distributed computation. In our design, this state size can be significantly reduced using the Oracle batching protocol. In particular, for a circuit represented

by a matrix of $N \times M$ elements, the state size will be proportional to one column, i.e. N elements. In this aspect, we obtain some properties of end-to-end IVC schemes [Sou23].

PCD from solely symmetric-key assumptions. The authors of [BMNW24a] built a bounded depth accumulator without using a proximity test as in [COS20] and without public key assumptions. Our work improves this result and shows that building a full-fledged split accumulator in this setting is possible.

A note on concurrent work. Our results were first presented at zkSummit12 [KNS24]. Soon after, two independent preprints, [BMNW24b] and [Sze24], were published. Both concurrent works use the idea of a STIR-based oracle batching protocol but for slightly different purposes.

The work of [Sze24] develops the DEEP Commitment notion, allowing for aggregating STARK proofs. The author considers the use case when knowledge extraction is not required.

The results of [BMNW24b] are more closely related to ours. Specifically, the authors introduce a many-to-one reduction for Reed–Solomon proximity claims, which is reminiscent of our oracle batching protocol. Building on this reduction, they present two accumulation schemes, one for Polynomial IOPs and one for R1CS.

In contrast, our work targets the construction of an accumulation scheme for correlated Holographic IOPs. As established in [BGK+23], this model captures a wide range of practical and performant proof systems based on univariate polynomials and proximity testing, including PLONK, Risc0, and Plonky2. Consequently, our results enable security level estimations for accumulation schemes instantiated over such systems with essentially no additional analysis.

Furthermore, we present concrete optimizations designed to minimize prover time and reduce the size of the accumulator. In particular, we demonstrate that for practical parameters of Holographic IOPs, the accumulator size can be orders of magnitude smaller than the witness size.

Finally, another contribution that distinguishes our work is the provision of a proof-of-concept implementation together with experimental evaluations. This demonstrates not only the theoretical but also the practical importance of our constructions.

2 Preliminaries

2.1 Reed–Solomon codes

Let $\mathbb{F}$ be a finite field, and $D \subset \mathbb{F}$. Given a function $f : D \to \mathbb{F}$, we denote by $\hat{f}$ the lowest-degree polynomial in $\mathbb{F}[X]$ that extends f.

Let $\mathsf{RS}[D, d]$ be the *Reed–Solomon code* over $D \subset \mathbb{F}$ with degree bound $d \mid \#D$, that is,

$$\mathsf{RS}[D, d] = \{f : D \to \mathbb{F} \mid \deg \hat{f} < d\}.$$

We might write $\mathsf{RS}[\mathbb{F}, D, d]$ if we want to make the field explicit, but most of the time we ignore it for simplicity. The *code rate* is defined as $\rho = d/\#D$.

Given $f, g : D \to \mathbb{F}$, we denote by $\Delta(f, g)$ the *relative Hamming distance* between them , i.e.

$$\Delta(f, g) = \frac{\#\{x \in D \mid f(x) \neq g(x)\}}{\#D}.$$

Similarly, for a set $S \subset \mathbb{F}^D$, we denote

$$\Delta(f, S) = \min_{g \in S}\{\Delta(f, g)\}.$$

We define the *list decoding* of a function $f : D \to \mathbb{F}$ as

$$\mathsf{List}(f, d, \delta) = \{g \in \mathsf{RS}[D, d] \mid \Delta(f, g) \leq \delta\}.$$

We say that $\mathsf{RS}[D, d]$ is (δ, ℓ)-*list decodable* if

$$\mathsf{List}(f, d, \delta) \leq \ell \qquad \forall f : D \to \mathbb{F}.$$

Folding preserves correlated agreement For $\delta \geq 0$, we say that $f_1, \ldots, f_n : D \to \mathbb{F}$ have δ-*corellated agreement in* $\mathsf{RS}[D, d]$ if there exist
– a set $S \subset D$, with size $\#S/\#D \geq (1 - \delta)$, and
– codewords $g_1, \ldots, g_n \in \mathsf{RS}[D, d]$,
such that

$$f_{i|S} = g_{i|S}, \quad \forall i = 1, \ldots, n.$$

In particular, we have

$$\Delta(f_i, \mathsf{RS}[D, d]) < \delta, \quad \forall i = 1, \ldots, n.$$

Let $f_1, \ldots, f_n : D \to \mathbb{F}$ and $\alpha \in \mathbb{F}$. We define:

$$\mathsf{Fold}_\alpha(f_1, \ldots, f_n) = \sum_{i=1}^{n} f_i \alpha^i.$$

Note that it is defined over the same domain as each individual function.

Lemma 1 ([BCI+20], Theorems 4.1 and 5.1 and [ACFY24a], Theorem 4.1). *Let $d \in \mathbb{N}, f_1, \ldots, f_n : D \to \mathbb{F}, \rho = d/\#D, \delta \in (0, 1 - \sqrt{\rho})$. Define the error term*

$$\varepsilon_{\mathsf{fold}} = \varepsilon_{\mathsf{fold}}(d, \rho, \delta, n) = \begin{cases} \frac{(n-1)\cdot d}{\rho \cdot \#\mathbb{F}} & \text{if } 0 < \delta \leq \frac{1-\rho}{2}. \\ \frac{(n-1)\cdot d^2}{\#\mathbb{F} \cdot \left(2 \cdot \min\{1 - \sqrt{\rho} - \delta, \frac{\rho}{20}\}\right)^7} & \text{if } \frac{1-\rho}{2} < \delta < 1 - \rho. \end{cases}$$

Suppose that $f_1, \ldots, f_n$ do not have δ-corellated agreement in $\mathsf{RS}[D, d]$. Then

$$\Pr\left[\alpha \leftarrow \mathbb{F} : \Delta(\mathsf{Fold}_\alpha(f_1, \ldots, f_n), \mathsf{RS}[D, d]) \leq \delta\right] \leq \varepsilon_{\mathsf{fold}},$$

Out-of-domain sampling

Lemma 2 ([ACFY24a], Lemma 4.5). *Let* $f : D \to \mathbb{F}, d, s \in \mathbb{N}, \delta \in [0,1]$. *Define the error term*

$$\varepsilon_{\mathsf{out}} = \binom{\ell}{2} \cdot \left(\frac{d-1}{\#\mathbb{F} - \#D}\right)^s \leq \frac{d^s \cdot \ell^2}{2 \cdot (\#\mathbb{F} - \#D)^s}.$$

If $\mathsf{RS}[D,d]$ *is* (δ,ℓ)*-list decodable, then*

$$\Pr\left[x_1,\ldots,x_s \leftarrow \mathbb{F} \setminus D : \begin{array}{c} \exists u,u' \in \mathsf{List}(f,d,\delta) \text{ s. t.} \\ u \neq u' \wedge u(x_i) = u'(x_i) \\ \forall i = 1,\ldots,s \end{array}\right] \leq \varepsilon_{\mathsf{out}}.$$

Quotienting Let $f : D \to \mathbb{F}$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^q$, with $D \cap \boldsymbol{x} = \emptyset$. We define $\mathsf{Quotient}(f, \boldsymbol{x}, \boldsymbol{y}) : D \to \mathbb{F}$ as follows. Let $\hat{p} \in \mathbb{F}[X]$ such that $\deg p < q$ and $p(x_j) = y_j$ for $j = 1,\ldots,q$. Then

$$\mathsf{Quotient}(f, \boldsymbol{x}, \boldsymbol{y})(x) = \frac{f(x) - \hat{p}(x)}{\prod_{j=1}^{q}(x - x_j)}.$$

Lemma 3 ([ACFY24a], Lemma 4.4). *Let* $f : D \to \mathbb{F}, d \in \mathbb{N}, \delta \in (0,1), \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^q$ *with* $D \cap \boldsymbol{x} = \emptyset$ *and* $q < d$. *Suppose that for every* $u \in \mathsf{List}(f,d,\delta)$, *there exists* $j \in \{1,\ldots,q\}$ *such that* $\hat{u}(x_j) \neq y_j$. *Then*

$$\Delta(\mathsf{Quotient}(f, \boldsymbol{x}, \boldsymbol{y}), \mathsf{RS}[D, d-q]) > \delta.$$

Degree correction Let $f : D \to \mathbb{F}$ and $d, d^* \in \mathbb{N}, r \in \mathbb{F}$ with $0 \leq d \leq d^*$. We define $\mathsf{DegCor} : D \to \mathbb{F}$ as follows:

$$\mathsf{DegCor}(d^*, r, f, d)(x) = f(x) \cdot \left(\sum_{\ell=0}^{d^*-d} (rx)^\ell\right).$$

Lemma 4 ([ACFY24a], Lemma 4.13). *Let* $f : D \to \mathbb{F}, d, d^* \in \mathbb{N}, r \in \mathbb{F}$ *with* $0 \leq d \leq d^*$. *Let* $\rho = d^*/\#D, \delta \in (0, \min\{1 - \sqrt{\rho}, 1 - \rho - 1/\#D\})$. *Suppose that* $\Delta(f, \mathsf{RS}[D,d]) > \delta$. *Then*

$$\Pr\left[r \leftarrow \mathbb{F} : \Delta(\mathsf{DegCor}(d^*, r, f, d), \mathsf{RS}[D, d^*]) \leq \delta\right] \leq \varepsilon_{\mathsf{corr}},$$

where $\varepsilon_{\mathsf{corr}} = \varepsilon_{\mathsf{fold}}(d^*, \rho, \delta, d^* + 1 - d)$ *is the error term defined in Lemma 1.*

2.2 Merkle commitments

We recall Merkle tree based vector commitment schemes, following the syntax of [CY24]. Let $k \in \mathbb{N}$, and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$. Let Σ be an alphabet, and let $l \in \mathbb{N}$ be the length of the vector to commit. The *Merkle tree vector commitment scheme* $\mathsf{MT}[k, \Sigma, l]$ is composed by the following algorithms:

– MT.Commit: receives as input a message vector $\boldsymbol{m} \in \Sigma^l$, and computes a Merkle commitment $\mathsf{rt} \in \{0,1\}^k$ and corresponding opening trapdoor $\mathsf{td} \in \{0,1\}^{O(k \cdot l)}$.
– MT.Open: receives as input opening trapdoor td and a subset $I \subseteq [l]$, and computes an opening proof ap that authenticates the values at the locations in I.
– MT.Check: receives as input a Merkle commitment rt, subset $I \subseteq [l]$, claimed values $\boldsymbol{a} \in \Sigma^I$, and opening proof ap, and computes a bit indicating whether the opening proof ap authenticates $\boldsymbol{a}$ as values for the locations in I with respect to rt.

Merkle tree vector commitments are used in the BCS transformation, described below, to swap oracles to functions by short descriptions of those functions, allowing to obtain an IP from an IOP.

2.3 Interactive oracle proofs

We denote oracle access to a function f by $\boxed{f}$. The result of a query at a point x is denoted by $\boxed{f(x)}$.

We follow the definitions of Holographic IOPs from [COS20]. We consider *indexed* relations $\mathcal{R} = \{(\mathbb{i}, \mathbb{x}, \mathbb{w})\}$. In the context of verifiable computation, $\mathbb{i}$ is an index that defines the computation, $\mathbb{x}$ is the statement that contains the public inputs, and $\mathbb{w}$ is the witness that contains the secret inputs. In many modern general-purpose proof systems, these take polynomial form. We define the *indexed language* $\mathcal{L}_{\mathbb{i}} = \{\mathbb{x} \mid \exists \mathbb{w} \text{ s. t. } (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}\}$.

The work of [BGK+23] considers *oracle relations*, that is, relations in which $\mathbb{w}$ may contain some functions, and $\mathbb{x}$ contains oracles to those functions. This is convenient to frame proximity proofs as IOPs, so we follow this approach. Thus, all definitions below apply to both regular relations and oracle relations, unless it is specified otherwise.

A *Holographic Interactive Oracle Proof (HIOP)* for an indexed relation $\mathcal{R}$ is a tuple $\Pi = (\mathcal{I}, \mathcal{P}, \mathcal{V})$, where $\mathcal{I}$ is a PT algorithm, and $\mathcal{P}, \mathcal{V}$ are stateful PPT algorithms. The prover and verifier engage interactively. The record of communications between them is called a *transcript*, and we denote it by $\pi \leftarrow \langle \mathcal{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}), \mathcal{V}^{\mathcal{I}(\mathbb{i})}(\mathbb{x}) \rangle$. At the end, the verifier examines the transcript and outputs a bit. We denote this by $0/1 \leftarrow \mathcal{V}^{\mathcal{I}(\mathbb{i})}(\mathbb{x}, \pi)$. A HIOP $\Pi = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ has *perfect completeness* if, for all $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$,

$$\Pr\left[\pi \leftarrow \langle \mathcal{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}), \mathcal{V}^{\mathcal{I}(\mathbb{i})}(\mathbb{x}) \rangle : 1 \leftarrow \mathcal{V}^{\mathcal{I}(\mathbb{i})}(\mathbb{x}, \pi)\right] = 1.$$

Flavors of soundness We first introduce the regular notions of soundness, as presented in [COS20, Section 4].[1]

[1] The error terms in the following definitions are allowed to depend on $\mathbb{i}, \mathbb{x}$, but we often omit this for simplicity. We will make the dependency explicit when there is some ambiguity.

Definition 1. *A HIOP $\Pi = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ has* soundness error ε *if, for any PPT adversary $\tilde{\mathcal{P}}$, all $\mathbb{i}$ and all $\mathbb{x} \notin \mathcal{L}_{\mathbb{i}}$, we have that*

$$\Pr\left[\pi \leftarrow \langle \tilde{\mathcal{P}}(\mathbb{i}, \mathbb{x}), \mathcal{V}^{\mathcal{I}(\mathbb{i})}(\mathbb{x})\rangle : 1 \leftarrow \mathcal{V}(\mathbb{i}, \mathbb{x}, \pi)\right] < \varepsilon,$$

over the coin tosses of $\mathcal{V}$.

Definition 2. *A HIOP $\Pi = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ has* knowledge error κ *if there exists a PPT extractor* Ext *such that, for any PPT adversary $\tilde{\mathcal{P}}$ and all $\mathbb{i}, \mathbb{x}$, we have that*

$$\Pr\left[\pi \leftarrow \langle \tilde{\mathcal{P}}(\mathbb{i}, \mathbb{x}), \mathcal{V}^{\mathcal{I}(\mathbb{i})}(\mathbb{x})\rangle : 1 \leftarrow \mathcal{V}(\mathbb{i}, \mathbb{x}, \pi)\right]$$
$$- \Pr\left[\mathbb{w} \leftarrow \mathsf{Ext}^{\tilde{\mathcal{P}}}(\mathbb{i}, \mathbb{x}) : (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}\right] < \kappa.$$

We now introduce round-by-round soundness and round-by-round knowledge soundness. These are interesting properties to us, because they are preserved through the transformations presented below.

We follow the formulation of [BGK+23, Definitions 3.12 and 3.13].

Definition 3. *A public-coin HIOP Π for a relation $\mathcal{R}$ has* round-by-round soundness error $\varepsilon_{\mathsf{rbr}}$ *if, for all indices $\mathbb{i}$, there exists a set* $\mathsf{DoomedSet}$ *such that:*

1. $\mathbb{x} \notin \mathcal{L}_{\mathbb{i}} \implies (\mathbb{x}; \emptyset) \in \mathsf{DoomedSet}$.
2. $(\mathbb{x}; \tau) \in \mathsf{DoomedSet} \implies \mathcal{V}^{\mathcal{I}(\mathbb{i})}(\mathbb{x}; \tau) = 0$ *for any complete transcript τ.*
3. $(\mathbb{x}; \tau) \in \mathsf{DoomedSet} \implies \Pr_{c \leftarrow \$}[(\mathbb{x}; \tau || m || c) \notin \mathsf{DoomedSet}] < \varepsilon_{\mathsf{rbr}}$ *for any partial transcript τ and any prover message m.*

Definition 4. *A public-coin HIOP Π for a relation $\mathcal{R}$ has* round-by-round knowledge error κ_{rbr} *if there exists a PT extractor* Ext *such that, for all indices $\mathbb{i}$, there exists a set* $\mathsf{DoomedSet}$ *such that:*

1. $(\mathbb{x}; \emptyset) \in \mathsf{DoomedSet}$ *for all $\mathbb{x}$.*
2. $(\mathbb{x}; \tau) \in \mathsf{DoomedSet} \implies \mathcal{V}^{\mathcal{I}(\mathbb{i})}(\mathbb{x}; \tau) = 0$ *for any complete transcript τ.*
3. $(\mathbb{x}; \tau) \in \mathsf{DoomedSet} \wedge \Pr_{c \leftarrow \$}[(\mathbb{x}; \tau || m || c) \notin \mathsf{DoomedSet}] > \kappa_{\mathsf{rbr}} \implies$ $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, *where* $\mathbb{w} \leftarrow \mathsf{Ext}(\mathbb{i}, \mathbb{x}, \tau, m)$, *for any partial transcript τ and any prover message m.*

From IOPs to non-interactive arguments in the ROM An IOP can be seen as a generalization of both IPs and PCPs, both of which can be transformed into non-interactive arguments, via the Fiat–Shamir transformation [FS87,PS96] and the CS proofs construction [Mic00,Val08], respectively. Thus, it is natural that a combination of these techniques yields a generic transformation from IOPs to non-interactive arguments. This was formalized as the BCS transformation [BCS16]. We summarize its properties in the following result.

Theorem 1. *[BCS16,CMS19,COS20,BGK+23] Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ be a random oracle, and let $Q \in \mathbb{N}$ be a bound on the number of queries to $\mathcal{H}$. Let*

$(\mathcal{I}, \mathcal{P}, \mathcal{V})$ *be a HIOP for a regular (non-oracle) relation* $\mathcal{R} = \{(\mathtt{i}, \mathtt{x}, \mathtt{w})\}$*, with the following properties:*[2]

- ℓ*: proof length.*
- q*: number of queries to oracles provided by the prover (and oracles in* $\mathtt{x}$*, in the case of oracle relations).*
- $\varepsilon_{\mathsf{rbr}}$*: round-by-round soundness error.*
- κ_{rbr}*: round-by-round knowledge error.*

Then, there exists a PT transformation BCS *such that* $(\mathsf{I}_{\mathsf{BCS}}, \mathsf{P}_{\mathsf{BCS}}, \mathsf{V}_{\mathsf{BCS}}) := \mathsf{BCS}^{\mathcal{H}}(\mathcal{I}, \mathcal{P}, \mathcal{V})$ *is a non-interactive holographic proof system for* $\mathcal{R}$ *in the ROM, with the following properties:*

- *adaptive soundness error and knowledge error against* Q*-query adversaries:*

$$\varepsilon_{\mathsf{fs}}(Q, k) = Q\varepsilon_{\mathsf{rbr}} + 3(Q^2 + 1)/2^k,$$
$$\kappa_{\mathsf{fs}}(Q, k) = Q\kappa_{\mathsf{rbr}} + 3(Q^2 + 1)/2^k.$$

- *adaptive soundness error and knowledge error against* $Q - O(q \log(\ell))$*-query quantum adversaries:*

$$\varepsilon_{\mathsf{qfs}}(Q, k) = \Theta(Q \cdot \varepsilon_{\mathsf{fs}}),$$
$$\kappa_{\mathsf{qfs}}(Q, k) = \Theta(Q \cdot \kappa_{\mathsf{fs}}).$$

2.4 δ-correlated IOPs

A common strategy to build SNARKs is to combine a polynomial IOP with a proximity test for a Reed–Solomon code, and the BCS transformation. We recall the notion of δ-correlated HIOPs, introduced in [BGK+23], which provides a framework for some such polynomial IOPs, e.g. Plonk [GWC19b]. We start by considering a certain type of oracle relations relative to a fixed Reed–Solomon code.

Definition 5. *An oracle relation* $\mathcal{R}$ *is a* $(\mathbb{F}, D, d)$-polynomial oracle relation *if the oracles in statements in* $\mathcal{R}$ *correspond to codewords in* $\mathsf{RS}[\mathbb{F}, D, d]$*.*

In particular, we consider the following strict $(\mathbb{F}, D, d)$-polynomial oracle relation.

$$\mathsf{CoAgg}(\delta) = \left\{ \begin{pmatrix} \mathtt{i} \\ \mathtt{x} \\ \mathtt{w} \end{pmatrix} = \begin{pmatrix} \mathbb{F}, D, d, \delta, n \\ \boxed{f_1, \ldots, f_n} \\ f_1, \ldots, f_n \end{pmatrix} \mid \begin{array}{c} \delta, n > 0, f_i : D \to \mathbb{F}, \\ \Delta(f_i, \mathsf{RS}[\mathbb{F}, D, d]) \leq \delta \quad \forall i \in [n] \\ \text{(with } \delta\text{-correlated agreement)} \end{array} \right\}.$$

Additionally, let $\mathsf{OCoAgg}(\delta)$ be a function that works as follows. It receives as input $(\mathtt{i}, \mathtt{x})$, with δ being the proximity parameter in $\mathtt{i}$. Then, it outputs 1 if and only if $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathsf{CoAgg}(\delta)$.

Given a statement $\mathtt{x}$ and a (potentially partial) transcript τ, let $\boxed{\mathsf{F}(\mathtt{x}, \tau)}$ denote the set of oracles that have appeared so far in either of them. We denote the set of functions behind these oracles by $\mathsf{F}(\mathtt{x}, \tau)$.

[2] As with the error terms, ℓ, q also depend on $\mathtt{i}, \mathtt{x}$.

Definition 6. *Let $\delta \geq 0$. A HIOP $\Pi = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for a $(\mathbb{F}, D, d)$-polynomial oracle relation $\mathcal{R}$ is* δ-correlated *if the following hold:*

- *$\mathcal{V}$ has oracle access to $\mathsf{OCoAgg}(\delta)$.*
- *Let τ denote the transcript up to the last round of interaction. For the last round:*
  - *$\mathcal{V}$ sends $\zeta \leftarrow S \subset \mathbb{F}$ (or an extension of $\mathbb{F}$).*
  - *$\mathcal{P}$ sends evaluations of functions in $\mathsf{F}(\mathbb{x}, \tau)$.*
- *$\mathcal{V}$'s final check consists of the following:*
  - *Assert whether the evaluations sent by the prover in the final round are roots of a certain multivariate polynomial, determined from $\mathbb{i}, \mathbb{x}, \tau$.*
  - *Check that a set of maps*

$$\{\mathsf{Quotient}(f_i, x_{i,\zeta}, f_i(x_{i,\zeta})) \mid f_i \in \mathsf{F}(\mathbb{x}, \tau)\}_{i=1}^{r}$$

  has δ-correlated agreement in $\mathsf{RS}[D, d-1]$, using OCoAgg on their oracles.

In our construction, we will need to deal with slightly more general protocols. The case with $d' = d - 1$ below encompasses δ-correlated IOPs. We will later encounter protocols with $d' = d$.

Definition 7. *Let $\delta \geq 0$. A HIOP $\Pi = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for a $(\mathbb{F}, D, d, d')$-polynomial oracle relation $\mathcal{R}$ is called a* semi-δ-correlated HIOP *if the following hold:*

- *$\mathcal{V}$ has oracle access to $\mathsf{OCoAgg}(\delta)$.*
- *$\mathcal{V}$'s final check consists of the following:*
  - *Assert whether the evaluations sent by the prover in the final round are roots of a certain multivariate polynomial, determined from $\mathbb{i}, \mathbb{x}, \tau$.*
  - *Using OCoAgg, check δ-correlated agreement in $\mathsf{RS}[D, d']$ of a single set of oracles.*

From δ-correrlated IOPs to regular IOPs One of the main results from [BGK+23] is that one can turn a 0-correlated HIOP Π for $\mathcal{R}$ into a δ-correlated IOP for $\mathcal{R}$, and then combine it with a HIOP for the $\mathsf{CoAgg}(\delta)$ relation to produce a standard HIOP for $\mathcal{R}$. We summarize the properties of the transformation here.

Theorem 2 ([BGK+23], Theorem 4.6). *Consider the following HIOPs:*

- *Π_{CA} is a HIOP for $\mathsf{CoAgg}(\delta)$, with round-by-round soundness error $\varepsilon_{\mathsf{rbr}}^{\mathsf{CA}}$.*
- *$\Pi^{\mathsf{OCoAgg}(0)} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ is a 0-correlated HIOP for a $(\mathbb{F}, D, d)$-polynomial oracle relation $\mathcal{R}$, with:*
  - *round-by-round soundness error $\varepsilon_{\mathsf{rbr}}$.*
  - *round-by-round knowledge error κ_{rbr}*

  Let ρ be the code rate of $\mathsf{RS}[D, d]$, and let $\delta < 1 - \sqrt{\rho}$ and $\eta = 1 - \sqrt{\rho} - \delta > 0$.

Then, there exists a HIOP Π for $\mathcal{R}$ with the following properties:

– *Round-by-round soundness error:*

$$\varepsilon'_{\mathsf{rbr}}(\mathtt{i}) = \max\left\{\frac{\varepsilon_{\mathsf{rbr}}(\mathtt{i})}{2\eta\sqrt{\rho}}, \varepsilon^{\mathsf{CA}}_{\mathsf{rbr}}(\mathtt{i}_{\mathsf{CA}})\right\},$$

where $\mathtt{i}_{\mathsf{CA}} = (\mathbb{F}, D, d, \delta, n)$*, and* n *is the number of functions* f_i *involved in* $\mathcal{V}$*'s final check for* δ*-correlated agreement.*

– *Round-by-round knowledge error:*

$$\kappa'_{\mathsf{rbr}}(\mathtt{i}) = \max\left\{\frac{\kappa_{\mathsf{rbr}}(\mathtt{i})}{2\eta\sqrt{\rho}}, \varepsilon^{\mathsf{CA}}_{\mathsf{rbr}}(\mathtt{i}_{\mathsf{CA}})\right\},$$

where $\mathtt{i}_{\mathsf{CA}}$ *is the same as in the item above.*

Given a 0-correlated HIOP Π, the transformation can be instantiated by using as Π_{CA} any correlated agreement protocol, like the batch variants of FRI [BBHR18], STIR [ACFY24a] or WHIR [ACFY24b].[3] Moreover, one can just use the trivial check in which the verifier reads the whole function by querying the oracles at every position. Afterwards, the BCS transformation can be applied, yielding a non-interactive argument in the ROM.

2.5 Split accumulation in the ROM

We follow the split accumulation definitions from [BCL+21], adapted to our setting. Let $\mathcal{R} = \{(\mathsf{qi}, \mathsf{qx}, \mathsf{qw})\}$ be a relation. A *split accumulation scheme* for $\mathcal{R}$ is a tuple of algorithms $\mathsf{SA} = (\mathsf{I}, \mathsf{P}, \mathsf{V}, \mathsf{D})$ with the following syntax:

– $\mathsf{I}(\mathsf{qi})$ outputs the index-specific prover key pk, verifier key vk, and decider key dk.
– $\mathsf{P}\left(\mathsf{pk}, (\mathsf{qx}_i, \mathsf{qw}_i)_{i=1}^n, (\mathsf{acc}_j)_{j=1}^m\right)$ outputs an accumulator $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$, and a proof π_{acc} of correct accumulation. We consider $\mathsf{acc}.\mathbb{x}$ and $\mathsf{acc}.\mathbb{w}$ the short part and long part of the accumulator, respectively.
– $\mathsf{V}\left(\mathsf{vk}, (\mathsf{qx})_{i=1}^n, (\mathsf{acc}_j.\mathbb{x})_{j=1}^m, \mathsf{acc}.\mathbb{x}, \pi_{\mathsf{acc}}\right)$ outputs $0/1$. Note that it only accesses the short part of accumulators.
– $\mathsf{D}(\mathsf{dk}, \mathsf{acc})$ outputs $0/1$. Unlike V, the decider has access to a full accumulator.

A *split accumulation scheme in the ROM* is a split accumulation scheme in which P, V have access to the same random oracle $\mathcal{H}$.

Definition 8 (Completeness). *A split accumulation scheme in the ROM* $\mathsf{SA} = (\mathsf{I}, \mathsf{P}, \mathsf{V}, \mathsf{D})$ *has* perfect completeness *if for any* $\left(\mathsf{qi}, (\mathsf{qx}_i, \mathsf{qw}_i)_{i=1}^n, (\mathsf{acc}_j)_{j=1}^m\right)$ *and*

[3] In fact, one can see these protocols as HIOPs for CoAgg, with $n = 1$ in the non-batch case.

any random oracle $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$*, we have that*

$$\Pr\left[\begin{array}{c} (\mathsf{pk},\mathsf{vk},\mathsf{dk}) \leftarrow \mathsf{I}(\mathsf{qi}), \\ (\mathsf{acc},\pi_{\mathsf{acc}}) \leftarrow \mathsf{P}^{\mathcal{H}}\left(\begin{array}{c}\mathsf{pk},\\ (\mathsf{qx}_i,\mathsf{qw}_i)_{i=1}^n,\\ (\mathsf{acc}_j)_{j=1}^m\end{array}\right) \end{array} : \begin{array}{c} (\mathsf{qi},\mathsf{qx}_i,\mathsf{qw}_i) \in \mathcal{R} \quad \forall i \in [n] \\ 1 \leftarrow \mathsf{D}(\mathsf{dk},\mathsf{acc}_j) \quad \forall j \in [m] \\ \Downarrow \\ 1 \leftarrow \mathsf{V}^{\mathcal{H}}\left(\begin{array}{c}\mathsf{vk},(\mathsf{qx}_i)_{i=1}^n,\\ (\mathsf{acc}_j.\mathbb{x})_{j=1}^m,\\ \mathsf{acc}.\mathbb{x},\pi_{\mathsf{acc}}\end{array}\right) \\ 1 \leftarrow \mathsf{D}(\mathsf{dk},\mathsf{acc}) \end{array}\right] = 1.$$

Definition 9 (Knowledge soundness). *A split accumulation scheme in the ROM* $\mathsf{SA} = (\mathsf{I},\mathsf{P},\mathsf{V},\mathsf{D})$ *has* knowledge error κ *if there exists a PPT extractor* Ext *such that, for any PPT* $\tilde{\mathsf{P}}$*, any auxiliary input* ai*, and any random oracle* $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$*, we have that*

$$\Pr\left[\begin{array}{c} \left(\begin{array}{c}\mathsf{qi},(\mathsf{qx}_i)_{i=1}^n,\\ (\mathsf{acc}.\mathbb{x})_{j=1}^m,\\ \mathsf{acc},\pi_{\mathsf{acc}},r\end{array}\right) \leftarrow \tilde{\mathsf{P}}^{\mathcal{H}}(\mathsf{ai}) \\ \left(\begin{array}{c}(\mathsf{qw}_i)_{i=1}^n,\\ (\mathsf{acc}_j.\mathbb{w})_{j=1}^m\end{array}\right) \leftarrow \mathsf{Ext}^{\mathcal{H},\tilde{\mathsf{P}}}(\mathsf{ai},r) \end{array} : \begin{array}{c} 1 \leftarrow \mathsf{V}^{\mathcal{H}}\left(\begin{array}{c}\mathsf{vk},(\mathsf{qx}_i)_{i=1}^n,\\ (\mathsf{acc}_j.\mathbb{x})_{j=1}^m,\\ \mathsf{acc}.\mathbb{x},\pi_{\mathsf{acc}}\end{array}\right) \\ 1 \leftarrow \mathsf{D}(\mathsf{dk},\mathsf{acc}) \\ \Downarrow \\ (\mathsf{qi},\mathsf{qx}_i,\mathsf{qw}_i) \in \mathcal{R} \quad \forall i \in [n] \\ 1 \leftarrow \mathsf{D}(\mathsf{dk},\mathsf{acc}_j) \quad \forall j \in [m] \end{array}\right] \geq 1-\kappa,$$

where r *is the randomness used by* $\tilde{\mathsf{P}}$ *above, and* $\mathsf{acc}_j = (\mathsf{acc}_j.\mathbb{x}, \mathsf{acc}_j.\mathbb{w})$*.*

Split accumulators are particularly useful if we can build them for the verifier relation of a NARK for circuit satisfiability. More precisely, let $\mathcal{R} = \{(\mathbb{i},\mathbb{x},\mathbb{w})\}$ be the relation for circuit satisfiability, and let $\mathsf{ARG} = (\mathcal{I},\mathcal{P},\mathcal{V})$ be a NARK for $\mathcal{R}$, and let us write

$$\begin{aligned} \mathsf{qi} &= \mathbb{i}, \\ \mathsf{qx} &= (\mathbb{x}, \pi.\mathbb{x}), \\ \mathsf{qw} &= \pi.\mathbb{w}. \end{aligned}$$

Then, we define the relation $\mathcal{R}_{\mathcal{V}}$ such that

$$(\mathsf{qi},\mathsf{qx},\mathsf{qw}) \in \mathcal{R}_{\mathcal{V}} \iff 1 \leftarrow \mathcal{V}(\mathsf{vk}_{\mathsf{NARK}},\mathbb{x},(\pi.\mathbb{x},\pi.\mathbb{w}))$$

where $\mathsf{vk}_{\mathsf{NARK}}$ is the NARK verifier key obtained from $\mathcal{I}(\mathbb{i})$.

Suppose that there is a split accumulation scheme SA for $\mathcal{R}_{\mathcal{V}}$. Then, we can use ARG and SA to build PCD schemes [BCL+21, Theorem 5.3].

3 Oracle batching

3.1 The core interactive protocol

Suppose that we have n functions $f_1,\ldots,f_n : D \to \mathbb{F}$. The verifier has oracle access to them, and the prover claims that f_i is δ-close to a low-degree polynomial,

for all $i = 1, \ldots, n$. The following protocol will produce a function f_{new}, and reduce the n claims above to the single claim that f_{new} is δ-close to a low-degree polynomial. The function f_{new} is defined over some other domain D' such that $D \cap D' = \emptyset$. The domains D, D' can be chosen as the two cosets of degree 2^s of the group of roots of unity of degree 2^{s+1}.

In our description below, we start from $f_i \in \mathsf{RS}[D, d]$ (or close) for $i = 1, \ldots, n$, and end up with $f_{\mathsf{new}} \in \mathsf{RS}[D']$ (or close), where t is a parameter that determines the number of verifier queries. Because we want to iteratively apply this protocol to aggregate incoming sets of functions of degree d, we will apply some degree correction to f_{new}, to bring it back to degree d.

We describe the interactive protocol in Algorithm 1, and denote by $f_{\mathsf{new}} \leftarrow \mathsf{OB}(f_1, \ldots, f_n)$ its execution with input oracles to $f_1, \ldots, f_n$ and output an oracle to f_{new}.[4]

Algorithm 1 The oracle batching (OB) interactive protocol

1: The Verifier samples $\alpha \leftarrow \mathbb{F}$
2: The Prover computes $f_{\mathsf{fold}} = \hat{\mathsf{Fold}}_\alpha(f_1, \ldots, f_n)_{|D'}$ and sends oracle $\boxed{f_{\mathsf{fold}}}$
3: The Verifier samples $x_1^{\mathsf{out}}, \ldots, x_s^{\mathsf{out}} \leftarrow \mathbb{F} \setminus D'$
4: The Prover sends $y_j^{\mathsf{out}} = \hat{f}_{\mathsf{fold}}(x_j^{\mathsf{out}}) \quad \forall j = 1, \ldots, s$
5: The Verifier samples $x_1, \ldots, x_t \leftarrow D, r \leftarrow \mathbb{F}$ and defines

$$\boxed{f_{\mathsf{quot}}} = \mathsf{Quotient}(\boxed{f_{\mathsf{fold}}}, \boldsymbol{x}, \boldsymbol{y}), \text{ where}$$
$$\boldsymbol{x} = (x_1^{\mathsf{out}}, \ldots, x_s^{\mathsf{out}}, x_1 \ldots, x_t) \text{ and } \boldsymbol{y} = (y_1^{\mathsf{out}}, \ldots, y_s^{\mathsf{out}}, y_1 \ldots, y_t)$$
$$y_j = \sum_{i=1}^{n} \boxed{f_i(x_j)} \cdot \alpha^i \quad \forall j = 1, \ldots, t$$
$$\boxed{f_{\mathsf{new}}} = \mathsf{DegCor}(d, r, \boxed{f_{\mathsf{quot}}}, d - (s+t))$$

Proposition 1. *Let $f_1, \ldots, f_n : D \to \mathbb{F}$, and let $d \in \mathbb{N}$. Suppose that, for all $i = 1, \ldots, n$, we have that $f_i \in \mathsf{RS}[D, d]$. Then*

$$\Pr\left[f_{\mathsf{new}} \leftarrow \mathsf{OB}(f_1, \ldots, f_n) : f_{\mathsf{new}} \in \mathsf{RS}[D', d]\right] = 1.$$

Proof. We need to show that $\deg \hat{f}_{\mathsf{quot}} < d - (s+t)$. From this, it will follow that $\deg \hat{f}_{\mathsf{new}} < d$, since DegCor is simply multiplying $\hat{f}_{\mathsf{quot}}$ by a polynomial of degree $t + s$.

To prove this, observe that, if $\deg \hat{f}_i < d$, then clearly $\deg \hat{f}_{\mathsf{fold}} < d$, as folding honestly does not increase the degree. Next, we show that the quotient behaves

[4] Note that $\mathsf{Quotient}(f_{\mathsf{fold}}, \boldsymbol{x}, \boldsymbol{y})$ is well defined, since f_{fold} is defined over D', whereas $x_1^{\mathsf{out}}, \ldots, x_s^{\mathsf{out}} \in \mathbb{F} \setminus D', x_1, \ldots, x_t \in D$, and $D \cap D' = \emptyset$.

as expected. Indeed, it is easy to check by inspection that $\hat{f}_{\mathsf{fold}}(x_j) = y_j$, and thus

$$\prod_{j=0}^{t}(X - x_j) \cdot \prod_{j=0}^{s}(X - x_j^{\mathsf{out}}) \mid \hat{f}_{\mathsf{fold}}(X) - \hat{p}(X),$$

where $\hat{p}$ is as defined in Section 2.1. Therefore, $\deg \hat{f}_{\mathsf{quot}} = \deg \hat{f}_{\mathsf{fold}} - (s+t) < d - (s+t)$, concluding the proof. □

3.2 A proximity proof from oracle batching

Given the oracle batching protocol above, $\mathsf{OB} = (\mathcal{P}_{\mathsf{OB}}, \mathcal{V}_{\mathsf{OB}})$, it is trivial to turn it into an IOP $\Pi_{\mathsf{OB}} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for the $\mathsf{CoAgg}(\delta)$ relation, i.e. a batched proximity check:

- $\mathcal{I}$ chooses the parameters of a Reed–Solomon code $\mathsf{RS}[\mathbb{F}, D, d]$, the proximity parameter δ, and the number n of functions in correlated agreement.
- $\mathcal{P}$ is the same as prover $\mathcal{P}_{\mathsf{OB}}$.
- $\mathcal{V}$ runs $\mathcal{V}_{\mathsf{OB}}$, and then checks directly whether $\Delta(f_{\mathsf{new}}, \mathsf{RS}[D', d]) \leq \delta$, accepting if and only if this check passes.

By itself, this protocol is not very useful because of the expensive verifier, but can fit nicely into the accumulator construction, where the expensive part of checking f_{new} is relegated to the decider.

Proposition 2. *Let $\delta \in (0, \min\{1-\sqrt{\rho}, 1-\rho-1/\#D\})$. The IOP Π_{OB} described above has round-by-round soundness error*

$$\varepsilon_{\mathsf{rbr}} = \max\{\varepsilon_{\mathsf{fold}}, \varepsilon_{\mathsf{out}}, \varepsilon_{\mathsf{corr}}, (1-\delta)^t\},$$

where $\varepsilon_{\mathsf{fold}}, \varepsilon_{\mathsf{out}}, \varepsilon_{\mathsf{corr}}$ are as defined in Lemmas 1, 2 and 4, respectively, t is the query parameter in the protocol, and δ is specified on $\mathtt{i}$.

Proof. We start by fixing an index $\mathtt{i}$, which fixes a certain Reed–Solomon code $\mathsf{RS}[D, d]$. Let $\mathcal{L}_{\mathtt{i}}$ be the language of statements for this particular $\mathtt{i}$. We want to prove that there exists $\mathsf{DoomedSet}$ in the conditions of Definition 3. We construct $\mathsf{DoomedSet}$ to include the following four types of entries.

(A) $(\mathtt{x}; \emptyset)$ such that $\mathtt{x} \notin \mathcal{L}_{\mathtt{i}}$, i.e. $f_1, \ldots, f_n$ do not have δ-correlated agreement.
(B) $(\mathtt{x}; \alpha)$ such that:
  - $\mathtt{x}$ is as in (A), and
  - α does not lead to an *unlucky fold*. An unlucky fold is the event that $\Delta(\mathsf{Fold}_\alpha(f_1, \ldots, f_n), \mathsf{RS}[D, d]) \leq \delta$.

(C) $(\mathtt{x}; \alpha || f_{\mathsf{fold}} || x_1^{\mathsf{out}}, \ldots, x_s^{\mathsf{out}})$ such that:
  - $(\mathtt{x}; \alpha)$ is as in (B), and
  - $x_1^{\mathsf{out}}, \ldots, x_s^{\mathsf{out}}$ do not lead to an *unlucky out-of-domain check*. An unlucky out-of-domain check is the event that $\exists u, u' \in \mathsf{List}(f_{\mathsf{fold}}, d, \delta)$ such that $u \neq u'$ and $u(x_j^{\mathsf{out}}) = u'(x_j^{\mathsf{out}})$ for all $j = 1, \ldots, s$.

(D) $(\mathbb{x}; \alpha || f_{\mathsf{fold}} || x_1^{\mathsf{out}}, \ldots, x_s^{\mathsf{out}} || y_1^{\mathsf{out}}, \ldots, y_s^{\mathsf{out}} || x_1, \ldots, x_t)$ such that:
- $(\mathbb{x}; \alpha || f_{\mathsf{fold}} || x_1^{\mathsf{out}}, \ldots, x_s^{\mathsf{out}})$ is as in (C), and
- $x_1, \ldots, x_t$ do not lead to an *unlucky spot check*. An unlucky spot check is the event that $\exists u \in \mathsf{List}(f_{\mathsf{fold}}, d, \delta)$ such that

$$\begin{aligned} \hat{u}(x_j^{\mathsf{out}}) &= y_j^{\mathsf{out}} \quad \forall j = 1, \ldots, s, \\ \hat{u}(x_j) &= y_j \quad \forall j = 1, \ldots, t. \end{aligned}$$

(E) $(\mathbb{x}; \alpha || f_{\mathsf{fold}} || x_1^{\mathsf{out}}, \ldots, x_s^{\mathsf{out}} || y_1^{\mathsf{out}}, \ldots, y_s^{\mathsf{out}} || x_1, \ldots, x_t || r)$ such that:
- $(\mathbb{x}; \alpha || f_{\mathsf{fold}} || x_1^{\mathsf{out}}, \ldots, x_s^{\mathsf{out}} || y_1^{\mathsf{out}}, \ldots, y_s^{\mathsf{out}} || x_1, \ldots, x_t)$ is as in (D), and
- r does not lead to an *unlucky degree correction*. An unlucky degree correction is the event that $\Delta(f_{\mathsf{new}}, \mathsf{RS}[D', d]) \leq \delta$.

Clearly, $\mathsf{DoomedSet}$ satisfies condition 1 from Definition 3, due to the inclusion of type-(A) entries.

We argue that full transcripts in $\mathsf{DoomedSet}$, i.e. type-(E) entries, are always rejected. Indeed, because an unlucky degree correction does not happen,

$$\Delta(f_{\mathsf{new}}, \mathsf{RS}[D', d]) > \delta,$$

so the direct check on f_{new} will fail, and the verifier will reject. Therefore, condition 2 is also met. It just remains to argue that condition 3 is met, that is, doomed transcripts stay doomed with high probability.

- Type-(A) transcripts become type-(B) transcripts unless we have an unlucky fold, which happens with probability $\varepsilon_{\mathsf{fold}}$, due to Lemma 1.
- Type-(B) transcripts become type-(C) transcripts unless we have an unlucky out-of-domain check, which happen with probability $\varepsilon_{\mathsf{out}}$, due to Lemma 2.
- Type-(C) transcripts become type-(D) transcripts unless we have an unlucky spot check, so we consider the probability of this event. Suppose that we have an unlucky spot check, i.e. $\exists u \in \mathsf{List}(f_{\mathsf{fold}}, d, \delta)$ such that

$$\begin{aligned} \hat{u}(x_j^{\mathsf{out}}) &= y_j^{\mathsf{out}} \quad \forall j = 1, \ldots, s, \\ \hat{u}(x_j) &= y_j \quad \forall j = 1, \ldots, t. \end{aligned}$$

  At this point, observe that:
  (i) An unlucky out-of-domain check did not happen, so u is unique.
  (ii) An unlucky fold did not happen, so $\Delta(\mathsf{Fold}_\alpha(f_1, \ldots, f_n), \mathsf{RS}[D, d]) > \delta$.
  Therefore, due to (i), the probability of an unlucky spot check happening is bounded by

$$\Pr\left[\begin{array}{c} \alpha \leftarrow \mathbb{F}, \\ x_1, \ldots, x_t \leftarrow D \end{array} : \hat{\mathsf{Fold}}_\alpha(f_1, \ldots, f_n)(x_j) = \hat{u}(x_j) \quad \forall j = 1, \ldots, t\right].$$

  Moreover, (ii) implies that $\mathsf{Fold}_\alpha(f_1, \ldots, f_n)$ and $\hat{u}_{|D}$ only agree on a $(1-\delta)$-fraction of the points in D at most, and therefore the probability of them agreeing on t random points $x_1, \ldots, x_t \leftarrow D$ is at most $(1-\delta)^t$.

– Type-(D) transcripts become type-(E) transcripts unless we have an unlucky degree correction. Because an unlucky spot check does not happen, for any $u \in \mathsf{List}(f_{\mathsf{fold}}, d, \delta)$, either:
  - there exists $j \in \{1, \ldots, s\}$ such that $\hat{u}(x_j^{\mathsf{out}}) \neq \hat{\mathsf{Fold}}_\alpha(f_1, \ldots, f_n)(x_j)$, or
  - there exists $j \in \{1, \ldots, t\}$ such that $\hat{u}(x_j) \neq \mathsf{Fold}_\alpha(f_1, \ldots, f_n)(x_j)$.

  Hence, by Lemma 3,

$$\Delta(f_{\mathsf{quot}}, \mathsf{RS}[D', d - (s+t)]) > \delta.$$

  Therefore, aplying Lemma 4 to $f_{\mathsf{new}} = \mathsf{DegCor}(d, r, f_{\mathsf{quot}}, d-(s+t))$, we have that

$$\Delta(f_{\mathsf{new}}, \mathsf{RS}[D', d]) > \delta,$$

  except with probability $\varepsilon_{\mathsf{corr}}$. This completes the proof.

□

Remark 1. The out-of-domain checks are necessary to reduce the probability of an unlucky spot check to the probability of $\mathsf{Fold}_\alpha(f_1, \ldots, f_n)$ agreeing on t random in-domain points with a *single* codeword u. Without this requirement, we could have different $u \in \mathsf{List}(f_{\mathsf{fold}}, d, \delta)$ that agree with $\mathsf{Fold}_\alpha(f_1, \ldots, f_n)$ in different subsets of D.

4 NARKs from correlated HIOP

In this section, we often deal with non-interactive versions of protocols obtained using the BCS transform or its modifications. In this case, the prover computes the Merkle-tree root for every oracle message and uses it as a short commitment For the sake of clarity, we will use $\boxed{\boxed{f}}$ as shorthand for the part of the output of the non-interactive prover corresponding to a (possibly virtual) oracle $\boxed{f}$. In particular, $\boxed{\boxed{f}}$ contains

– The Merkle-tree root $\mathsf{rt} = \mathsf{MT.Commit}(\mathsf{g}|_{\mathsf{D}})$ for some function $g : D \to \mathbb{F}$. If $\boxed{f}$ is not virtual then $g = f$,
– (d^*, r, d) if $\boxed{f} = \mathsf{DegCor}(d^*, r, \boxed{g}, d)$,
– $(\boldsymbol{x}, \boldsymbol{y})$ if $\boxed{f} = \mathsf{Quotient}(\boxed{g}, \boldsymbol{x}, \boldsymbol{y})$.

We also denote by $\mathtt{Check}$ the verification that all **non-empty** value-position pairs of $(v_i, \mathsf{pos}_i)_{i=1}^t$ with the corresponding Merkle-tree authentication paths $(\mathsf{ap}_i)_{i=1}^t$ are consistent with the description of the oracle.

$\mathtt{Check}((v_i, \mathsf{pos}_i)_{i=1}^t, (\mathsf{ap}_i)_{i=1}^t, \boxed{\boxed{f}}) \to 0/1$:
1. for $i = 1, \ldots, t$:
2. Parse value $g_i = g(\mathsf{pos}_i)$ from ap_i.
3. Given g_i and $\boxed{\boxed{f}}$, calculate $f(\mathsf{pos}_i)$.

4. $b_i = \Big((f(\mathsf{pos}_i) = v_i) \wedge (\mathsf{MT.Check}(\boxed{f}.\mathsf{rt}, \mathsf{pos}_i, g_i, \mathsf{ap}_i) = 1)\Big) \vee$
$\vee\, ((v_i = \perp) \wedge (\mathsf{ap}_i = \perp))$.
5. **return** $\wedge_{i=1}^{t} b_i$.

To begin, we present a slight modification of transformation that allows us to compile a δ-correlated HIOP and IOPP into a classical HIOP (Theorem 2). The main difference is that we allow as an input a semi-δ-correlated HIOP (Definition 7) and split the Verifier into two parts. The first one makes his final decision before the interactive part of the second one starts. This way, we can separate the part of the proof (in the non-interactive version) for independent verification. This transformation simplifies our construction of the split accumulator. Let $\delta \geq \delta_0$.

Definition 10. *The transformation* $\mathsf{T}[\delta, \delta_0]$ *takes as input a public coin generalized* δ*-correlated HIOP* $\Pi = (\mathsf{I}, \mathsf{P}, \mathsf{V})$ *for indexed polynomial oracle relation* $\mathcal{R} = (\mathtt{i}, \mathbb{x}, \mathbb{w})$*, HIOP* $\Pi_{\mathsf{CA}} = (\mathsf{I}_{\mathsf{CA}}, \mathsf{P}_{\mathsf{CA}}, \mathsf{V}_{\mathsf{CA}})$ *for polynomial oracle relation* $\mathsf{CoAgg}(\delta_0)$ *and outputs a plain HIOP* $(\mathcal{I}, \mathcal{P}, \mathcal{V})$ *defined below.*

- $\mathcal{I}$ *outputs encodings* $\Pi.\mathsf{I}(\mathtt{i})$ *and* $\mathtt{i}_{\mathsf{CA}}(\mathtt{i})$*, where the latter contains the parameters of a Reed–Solomon code* $\mathsf{RS}[F, D, d]$*, the proximity parameter* δ_0*, and the number* n *of functions in correlated agreement.*
- $\mathcal{P}(\mathtt{i}, \mathbb{x}, \mathbb{w})$ *is a pair of interactive algorithms* $\mathcal{P}_1, \mathcal{P}_2$*, where*
  - $\mathcal{P}_1$ *is the same as the prover* $\Pi.\mathsf{P}$*,*
  - $\mathcal{P}_2$ *is the same as the prover* $\Pi_{\mathsf{CA}}.\mathsf{P}_{\mathsf{CA}}$*.*

  First, the algorithm $\mathcal{P}_1(\mathtt{i}, \mathbb{x}, \mathbb{w})$ *is executed. Let* $\boldsymbol{f} = (f_1, \ldots, f_n)$ *be the words on which the verifier would call the oracle* $\mathsf{OCoAgg}(\delta)$ *during its final decision process. Then, the algorithm* $\mathcal{P}_2(\mathtt{i}_{\mathsf{CA}}, \boxed{\boldsymbol{f}}, \boldsymbol{f})$ *is executed.*
- $\mathcal{V}^{\mathcal{I}(\mathtt{i})}(\mathbb{x})$ *is a pair of interactive algorithms* $\mathcal{V}_1, \mathcal{V}_2$*, where*
  - $\mathcal{V}_1$ *is the same as the verifier* $\Pi.\mathsf{V}$ *except that it does not call oracle* OCoAgg*. Instead, upon completion, it sends an additional "dummy" message.*
  - $\mathcal{V}_2$ *is the same as the verifier* $\Pi_{\mathsf{CA}}.\mathsf{V}_{\mathsf{CA}}$*.*

  Verifier launches $\mathcal{V}_1^{\mathcal{I}(\mathtt{i})}(\mathbb{x})$*, then* $\mathcal{V}_2^{\mathtt{i}_{\mathsf{CA}}}(\boxed{\boldsymbol{f}})$*.* $\mathcal{V}$ *accepts if and only if* $\mathcal{V}_1 \wedge \mathcal{V}_2$*.*

To obtain the corresponding non-interactive algorithm, we introduce a modification of the BCS transformation that reflects the two-component format of the verifier obtained as a result of the T transformation.

Definition 11. *The transformation* BCST *takes as input a public coin HIOP* $\Pi' = \mathsf{T}[\delta, \delta_0](\Pi, \Pi_{\mathsf{CA}})$ *and outputs the preprocessing non-interactive argument in the ROM* $(\mathbf{I}, \mathbf{P}, \mathbf{V} = (\mathbf{V_1}, \mathbf{V_2}))$*, defined below.*

- $\mathbf{I}^{\mathcal{H}}(\mathtt{i}) \to (\mathsf{pk}, \mathsf{vk})$*. The algorithm is the same as what we would get by applying the BCS transformation. For simplicity, we write the* $\mathtt{i}_{\mathsf{CA}}$ *parameters directly into the verification* vk *and proving* pk *keys. We denote corresponding deterministic extraction algorithm as* $\mathtt{Parse}(\mathsf{vk}/\mathsf{pk})) \to \mathtt{i}_{\mathsf{CA}}$*.*
- $\mathbf{P}^{\mathcal{H}}(\mathsf{pk}, \mathbb{x}, \mathbb{w}) \to \pi = (\pi.\mathbb{x}, \pi.\mathbb{w})$*. We make the following changes compared to the prover obtained using the* BCS *transform.*

- *If the prover sends several oracles in one round, a separate commitment is calculated for each in the non-interactive version.*
- *Non-oracle values are written directly to the proof. These include polynomial evaluations provided by the prover, parameters for calculating virtual oracle values, etc.*
- *The output is divided into two parts, corresponding to the execution of $\Pi'.\mathcal{P}_1$ and $\Pi'.\mathcal{P}_2$ respectively. The intermediate root hash σ_{k_1} is written to the first part of the proof so that $\Pi'.\mathcal{V}_1$ can perform the check independently of the second part of the proof.*

Then the proof π has the form

$$\pi.\mathbb{x} = \left((\mathsf{m}_1, \ldots, \mathsf{m}_{k_1}), (\mathsf{ap}_1, \ldots, \mathsf{ap}_{q_1}), \sigma_{k_1}\right),$$
$$\pi.\mathbb{w} = \left((\mathsf{m}^{\mathsf{CA}}_1, \ldots, \mathsf{m}^{\mathsf{CA}}_{k_2}), (\mathsf{ap}^{\mathsf{CA}}_1, \ldots, \mathsf{ap}^{\mathsf{CA}}_{q_2}), \sigma^{\mathsf{CA}}_{k_2}\right),$$

where k_1 and k_2 are the number of oracle messages sent by $\Pi'.\mathcal{P}_1$ and $\Pi'.\mathcal{P}_2$, respectively, q_1 and q_2 are the number of requests to these oracles and the index made by $\Pi'.\mathcal{V}_1$ and $\Pi'.\mathcal{V}_2$, respectively. Messages $\mathsf{m}_i, \mathsf{m}^{\mathsf{CA}}_j$ contain the roots of Merkle-trees and non-oracle values.

– $\mathbf{V}^{\mathcal{H}}(\mathsf{vk}, \mathbb{x}, \pi) = (\mathbf{V}^{\mathcal{H}}_{\mathbf{1}}, \mathbf{V}^{\mathcal{H}}_{\mathbf{2}})(\mathsf{vk}, \mathbb{x}, \pi) \to \{0, 1\}$. *We split the verifier that we obtained using the BCS transformation into two parts.*
  - $\mathbf{V}^{\mathcal{H}}_{\mathbf{1}}(\mathsf{vk}, \mathbb{x}, \pi.\mathbb{x})$ *verifies the first part of the proof, makes a decision, and saves the state* $\mathsf{aux} = (\boxed{f_1, \ldots, f_n}, \sigma_{k_1})$. *We denote the deterministic oracle extraction algorithm as* $\mathtt{ParseDesc}(\mathsf{vk}, \pi.\mathbb{x})$.
  - $\mathbf{V}^{\mathcal{H}}_{\mathbf{2}}(\mathsf{aux}, \mathsf{vk}, \mathbb{x}, \pi.\mathbb{w})$ *uses the state to continue verifying the rest of the proof. In particular, it checks the consistency of the Merkle-tree paths provided in the proof with the corresponding* $\boxed{f_i}$.

  $\mathbf{V}$ *accepts if and only if*
  1. $\mathbf{V}^{\mathcal{H}}_{\mathbf{1}}(\mathsf{vk}, \mathbb{x}, \pi.\mathbb{x}) = 1$
  2. $\mathbf{V}^{\mathcal{H}}_{\mathbf{2}}(\mathsf{aux}, \mathsf{vk}, \mathbb{x}, \pi.\mathbb{w}) = 1$, *where* $\mathsf{aux} = (\mathtt{ParseDesc}(\mathsf{vk}, \pi.\mathbb{x}), \sigma_{k_1})$

Now, let us consider a trivial version of the proximity test. Although such a test is of no practical interest, it helps us conceptualize the split accumulation scheme. Let $\Pi_{\mathsf{TRV}} = (\mathsf{I}, \mathsf{P}, \mathsf{V})$ is a HIOP for the polynomial oracle relation $\mathsf{CoAgg}(\delta)$ (Section 2.4).

– I outputs $\mathbb{i}_{\mathsf{CA}}$, which contains the parameters of a Reed–Solomon code $\mathsf{RS}[F, D, d]$, the proximity parameter δ_0, and the number n of functions in correlated agreement.
– V checks directly (by reading the oracle $\boxed{\boldsymbol{f}}$ entirely) whether $(\mathbb{i}_{\mathsf{CA}}, \boxed{\boldsymbol{f}}, \boldsymbol{f}) \in \mathsf{CoAgg}(\delta_0)$, accepting if this check passes.

Suppose that we use the transformation T together with the HIOP Π_{TRV} for some δ_0-correlated HIOP Π. Then, in the non-interactive version of the result

$$\mathsf{BCST}(\mathsf{T}[\delta, \delta_0](\Pi, \Pi_{\mathsf{TRV}})),$$

the second part of the proof $\pi.\mathbb{w}$ will contain only value-authentication path pairs $(v_i, \mathsf{ap}_i^{\mathsf{CA}})_{i=1}^{n\cdot|D|}$. This is a consequence of the fact that the prover $\Pi_{\mathsf{TRV}}.\mathsf{P}$ does not send additional oracle messages and the verifier $\Pi_{\mathsf{TRV}}.\mathsf{V}$ does not need additional randomness. The job of the verifier $\mathbf{V_2}$ is

- to check the authentication paths ap_i with respect to the $\boxed{f_1, \ldots, f_n}$ contained in aux, and the oracle $\mathcal{H}$. We denote the deterministic values and authentication paths extraction algorithms as $\mathtt{ParseEval}(\mathsf{vk}, \pi.\mathbb{w}) \to v_i$ and $\mathtt{ParseAP}(\mathsf{vk}, \pi.\mathbb{w}) \to \mathsf{ap}_i$, respectively. Then:

  $\mathsf{CheckAP}(\boxed{f}, \mathsf{vk}, \pi.\mathbb{w}) \to 0/1$:
  1. $v_i \leftarrow \mathtt{ParseEval}(\mathsf{vk}, \pi.\mathbb{w})$
  2. $\mathsf{ap}_i \leftarrow \mathtt{ParseAP}(\mathsf{vk}, \pi.\mathbb{w})$
  3. **return** $\mathtt{Check}((v_i, i), (\mathsf{ap}_i), \boxed{f})$
- to check that the resulting vectors of values is δ_0-close to the $\mathsf{RS}[\mathbb{F}, D, d]$, and make the final decision.

Therefore, $\mathbf{V_2}$ only needs a simplified interface (without $\mathbb{x}$ and σ_{k_1}):

$$\mathbf{V}_{\mathbf{2}}^{\mathcal{H}}(\mathsf{aux}', \mathsf{vk}, \pi.\mathbb{w}), \text{ where } \mathsf{aux}' = \mathtt{ParseDesc}(\mathsf{vk}, \pi.\mathbb{x}).$$

Since we only check for proximity to the code, some values may differ from the expected ones. Therefore, we will also allow some authentication paths to be missing. To reflect this, we will introduce another modification of the BCS transformation (Definition 12) for the special case of Π_{TRV}.

Definition 12. *The transformation* $\mathsf{BCST}^{\perp}$ *takes as input a public coin HIOP* $\Pi' = \mathsf{T}[\delta, \delta_0](\Pi, \Pi_{\mathsf{TRV}})$ *and outputs the preprocessing non-interactive argument in the ROM* $(\mathbf{I}, \mathbf{P}, \mathbf{V} = (\mathbf{V_1}, \mathbf{V_2}))$. *The transformation* $\mathsf{BCST}^{\perp}$ *is identical to* BCST *except for the algorithm* $\mathbf{V_2}$*, which can now accept the symbol* $\perp$ *instead of the authentication path* ap_i*. In this case, when checking the proximity to the* $\mathsf{RS}[\mathbb{F}, D, d]$*, it considers the corresponding value to be* $\perp$*.*

Lemma 5. *Let* $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ *be a random oracle, and let* $Q \in \mathbb{N}$ *be a bound on the number of queries to* $\mathcal{H}$*. Let* Π_0 *be a semi-*δ*-correlated HIOP for an indexed regular (non-oracle) relation, and* $\Pi' = \mathsf{T}[\delta, \delta_0](\Pi_0, \Pi_{\mathsf{TRV}})$ *has round-by-round knowledge error* κ*. Then* $\Pi = \mathsf{BCST}(\Pi')$ *has knowledge error against* Q*-query adversaries*

$$Q\kappa + 3(Q^2 + 1)/2^k, \tag{1}$$

and $\Pi^{\perp} = \mathsf{BCST}^{\perp}(\Pi')$ *has knowledge error against* Q*-query adversaries*

$$Q\kappa + 3(Q^2 + 1)/2^k. \tag{2}$$

Proof. BCST is essentially the same as BCS applied to HIOPs of the form $\mathsf{T}[\delta, \delta_0](\Pi_0, \Pi_{\mathsf{CA}})$ only with non-essential semantic modifications. Theorem 1 is valid for the BCST transformation; therefore, the upper bound of the knowledge error (1) is its immediate consequence.

Before arguing for the bound (2), we briefly recall the construction of the BCS knowledge extractor, as defined in [BCS16]. It is constructed in two stages.

1. Given a prover $\widetilde{\mathbf{P}}$ for Π, a prover $\widetilde{\mathcal{P}}(\widetilde{\mathbf{P}})$ is constructed for the underlying HIOP Π'. $\widetilde{\mathcal{P}}$ is constructed so that its ability to cheat in a state restoration attack is closely related to $\widetilde{\mathbf{P}}$'s ability to cheat.
   Essentially, the prover $\widetilde{\mathcal{P}}$ runs $\widetilde{\mathbf{P}}$ and simulates responses for queries to a random oracle. Given a full query-response table, $\widetilde{\mathcal{P}}$ uses the Valiant's extractor to obtain partial openings that are consistent with the Merkle tree roots, fills in the tree's missing leaves with zeros and sends it to the interactive verifier.
   The verifier accepts except the probability that $\widetilde{\mathbf{P}}$ has found a collision for the Random Oracle and violated the intended order of calls. This probability of cheating by the prover is upper bounded by $3(Q^2+1)/2^k$.
2. The underlying HIOP Π' has the state restoration knowledge error

$$\varepsilon_{\mathsf{sr}} \leq \kappa \cdot Q.$$

   Thus we can use the corresponding extractor $\mathcal{E}_{\mathsf{sr}}$ with $\widetilde{\mathcal{P}}$ as an oracle to derive the witness.

Moving on to the upper bound for the knowledge error (2), we argue that given a prover $\widetilde{\mathbf{P}}^{\perp}$ for $\Pi^{\perp}$, an identical construction $\widetilde{\mathcal{P}}(\widetilde{\mathbf{P}}^{\perp})$ allows us to obtain a prover for the underlying HIOP Π' with the same complexity and probability of success as in the case above. Since Π and $\Pi^{\perp}$ have the same underlying HIOP Π' and hence the same state restoration knowledge error, the lemma statement will follow automatically.

Indeed, note that

- Valiant's extractor returns a partial opening corresponding to the roots of the Merkle trees; unknown elements are filled with zeros.
- When $\Pi^{\perp}.\mathbf{V_2}$ accepts, deterministic $\Pi_{\mathsf{TRV}}.\mathsf{V}$ also accepts when given oracles to the messages extracted by the Valiant's extractor because if a vector in $(\mathbb{F} \cup \{\perp\})^{\#D}$ is δ-close, then the vector obtained from it by replacing $\perp$ with 0 is certainly δ-close.

Thus, $\widetilde{\mathcal{P}}$ wins the state restoration attack if $\widetilde{\mathbf{P}}^{\perp}$ was successful, except for the probability that $\widetilde{\mathbf{P}}^{\perp}$ used a random oracle collision and violated the order of calls.

It remains to note that the probability of the prover's cheating associated with the random oracle is also upper bounded by $3(Q^2+1)/2^k$ (following Lemma 7.3 of the [BCS16]). □

Finally, let us consider the last ingredient. We denote by $\mathsf{OCoAgg}^{(\mathsf{k})}$ the oracle for the verification algorithm of the correlation agreement for a set of k functions. Then, we present a transformation from a δ-correlated HIOP, during which the verifier calls $\mathsf{OCoAgg}^{(\mathsf{n})}$, to a δ-correlated HIOPk during which the verifier calls $\mathsf{OCoAgg}^{(1)}$.

Definition 13. *The transformation* B *takes as input a public coin* δ*-correlated HIOP* $\Pi = (\mathsf{I}, \mathsf{P}, \mathsf{V}^{\mathsf{OCoAgg}^{(\mathsf{n})}})$ *for indexed polynomial oracle relation* $\mathcal{R} = (\mathtt{i}, \mathtt{x}, \mathtt{w})$

and outputs a semi-δ-correlated HIOP $\mathsf{B}(\Pi) = (\mathsf{I}_\mathsf{B}, \mathsf{P}_\mathsf{B}, \mathsf{V}_\mathsf{B}^{\mathsf{OCoAgg}^{(1)}})$ *for the same relation* $\mathcal{R}$*. Let* $\mathsf{OB} = (\mathcal{P}_\mathsf{OB}, \mathcal{V}_\mathsf{OB})$ *be as defined in Section 3.*

- I_B *is same as the indexer* I*,*
- $\mathsf{P}_\mathsf{B}(\mathtt{i}, \mathtt{x}, \mathtt{w})$ *is a pair of interactive algorithms* $\mathcal{P}_1, \mathcal{P}_2$*, where*
  - $\mathcal{P}_1$ *is the same as the prover* $\Pi.\mathsf{P}$*,*
  - $\mathcal{P}_2$ *is the same as the prover* $\mathsf{OB}.\mathcal{P}_\mathsf{OB}$ *.*

  First, the algorithm $\mathcal{P}_1(\mathtt{i}, \mathtt{x}, \mathtt{w})$ *is executed. Let* $\boldsymbol{f} = (f_1, \ldots, f_n)$ *be the words on which the verifier would call the oracle* $\mathsf{OCoAgg}^{(\mathsf{n})}(\delta)$ *during its final decision process. Then, the algorithm* $\mathcal{P}_2(\mathtt{i}_{\mathsf{CA}}, \boxed{\boldsymbol{f}}, \boldsymbol{f})$ *is executed.*
- $\mathsf{V}_\mathsf{B}^{\mathcal{I}(\mathtt{i}), \mathsf{OCoAgg}^{(1)}}(\mathtt{x})$ *is a pair of interactive algorithms* $\mathcal{V}_1, \mathcal{V}_2$*, where*
  - $\mathcal{V}_1$ *is the same as verifier* $\Pi.\mathsf{V}$ *except that it does not call oracle* OCoAgg*. Instead, upon completion, it sends an additional "dummy" message.*
  - $\mathcal{V}_2$ *is the same as verifier* $\mathsf{OB}.\mathcal{V}_\mathsf{OB}$*, except that, upon completion,* $\mathcal{V}_2$ *queries* $\mathsf{OCoAgg}^{(1)}(\boxed{f_{\mathsf{new}}})$*.*

  The verifier launches $\mathcal{V}_1^{\mathcal{I}(\mathtt{i})}(\mathtt{x})$*, then* $\mathcal{V}_2^{\mathtt{i}_{\mathsf{CA}}}(\boxed{\boldsymbol{f}})$*.* $\mathcal{V}$ *accepts if and only if* $\mathcal{V}_1 \wedge \mathcal{V}_2$*.*

The next lemma is a version of [BGK+23, Lemma 4.8] and its proof is verbatim.

Lemma 6. *Let* Π *be a RBR knowledge sound* δ*-correlated HIOP with knowledge error* κ_Π*. Then,* $\mathsf{HIOP} := \mathsf{T}[\delta, \delta_0](\mathsf{B}(\Pi), \Pi_{\mathsf{TRV}})$ *is a RBR knowledge sound HIOP with knowledge error* $\kappa = \max\{\kappa_\Pi, \epsilon_\mathsf{OB}\}$ *for all* $\delta_0 \leq \delta$*. Here,* ϵ_OB *is the RBR soundness error of* Π_OB*.*

Proof. The proof of [BGK+23, Lemma 4.8] works verbatim. For the sake of completeness, let us spell out the proof. (Before starting the proof, let us recall that Π_OB is the HIOP for the CoAgg-relation obtained from OB and Π_{TRV}.)

Recall that $\mathsf{B}(\Pi).\mathsf{P}$ is a successive execution of $\Pi.\mathsf{P}$ and $\mathsf{OB}.\mathsf{P}$, and $\mathsf{B}(\Pi).\mathsf{V}$ is a successive execution of "$\Pi.\mathsf{V}$ without OCoAgg-queries at the end", $\mathsf{OB}.\mathsf{V}$ and "OCoAgg-query for f_{new}". Therefore, the prover and the verifier of HIOP are

- $\mathsf{HIOP}.\mathsf{P} = (\Pi.\mathsf{P}, \mathsf{OB}.\mathsf{P}', \mathtt{send}\ \boxed{f_{new}})$
- $\mathsf{HIOP}.\mathsf{V} = (\Pi.\mathsf{V}', \mathsf{OB}.\mathsf{V}, \mathsf{OCoAgg}\ \mathtt{query\ for}\ \boxed{f_{new}})$,

where $\mathsf{OB}.\mathsf{P}'$ indicates OB.P without the first step of sending oracles to batch and $\Pi.\mathsf{V}'$ indicates $\Pi.\mathsf{V}$ without the final call to OCoAgg. (The Π-phase ends with a prover message, which is followed by a message α by $\mathsf{OB}.\mathsf{V}$.)

Let $\mathsf{DoomedSet}_\Pi$ be the doomed set relative to which Π is RBR knowledge sound. Let $\mathsf{DoomedSet}_\mathsf{OB}$ be the dommed set relative to which Π_OB is RBR sound. We claim that the following is the doomed relative to which HIOP is RBR knowledge sound:

$$\begin{aligned}\mathsf{DoomedSet}_{\mathsf{HIOP}} =& \mathsf{DoomedSet}_\Pi \cup \{(x, \tau, v, \alpha) : (x, \tau) \in \mathsf{DoomedSet}_\Pi\} \\ &\cup \{(x, \tau, v, \alpha, \tau') : \tau' \neq \emptyset, \Pi.\mathsf{V}'\ \mathtt{rejects}\ (x, \tau, v) \\ &\mathtt{and}\ (\boxed{f_1}, \ldots, \boxed{f_n}, \alpha, \tau') \in \mathsf{DoomedSet}_\mathsf{OB}\},\end{aligned}$$

where v is the Π-prover's final message, which consists of evaluations of polynomials, α is the first verifier message in OB and $\boxed{f_i}$ are oracles send by the Π-prover, which are part of τ.

By definition, all instance-only transcripts are in $\mathsf{DoomedSet}_{\mathrm{HIOP}}$. Since any complete transcript in $\mathsf{DoomedSet}_{\mathrm{HIOP}}$ contains a complete doomed Π_{OB}-transcript (which by the soundness of Π_{OB} is rejected by the Π_{OB}-verifier), OCoAgg rejects when queries for f_{new}. Hence HIOP.V rejects any doomed complete transcript.

Thus, it remains show the extractability when transcripts escape the doomed set with probability $> \kappa := \max\{\kappa_{\Pi}, \epsilon_{\mathsf{OB}}\}$. If the escape occurs before the Π-prover sends the final message, (i.e. strictly before the transcript reaches the full transcript for Π), by the RBR knowledge soundness of Π, a satisfying witness can be extracted.

The next case is when

$$(x, \tau) \in \mathsf{DoomedSet}_{\mathrm{HIOP}} \wedge \Pr_c[(x, \tau, v, c) \notin \mathsf{DoomedSet}_{\mathrm{HIOP}}] > \kappa.$$

However, this does not occur because $(x, \tau, v, c) \notin \mathsf{DoomedSet}_{\mathrm{HIOP}}$ if and only if $(x, \tau) \notin \mathsf{DoomedSet}_{\mathrm{HIOP}}$ by definition.

Next is the case that

$$(x, \tau, v, \alpha) \in \mathsf{DoomedSet}_{\mathrm{HIOP}} \wedge \Pr_c[(x, \tau, v, \alpha, \boxed{f_{fold}}, c) \notin \mathsf{DoomedSet}_{\mathrm{HIOP}}] > \kappa.$$

In case (x, τ, v) does not pass $\Pi.\mathsf{V}'$, $(x, \tau, v, \alpha, \boxed{f_{fold}}, c) \in \mathsf{DoomedSet}_{\mathrm{HIOP}}$ by definition. Thus, this case does not exist. Let us thus assume (x, τ, v) passes $\Pi.\mathsf{V}'$. By the definition of $\mathsf{DoomedSet}_{\mathrm{HIOP}}$, $(x, \tau, v, \alpha, \boxed{f_{fold}}, c) \notin \mathsf{DoomedSet}_{\mathrm{HIOP}}$ implies $(\{\boxed{f_i}\}_{i=1}^n, \alpha, \boxed{f_{fold}}, c) \notin \mathsf{DoomedSet}_{\mathsf{OB}}$. By the completeness of Π_{OB}, if $\{\boxed{f_i}\}_{i=1}^n$ has a δ-correlated agreement, this does not occur. Hence, we may assume that $\{\boxed{f_i}\}_{i=1}^n$ does not have a δ-correlated agreement. In this case, by the RBR knowledge soundness of $\mathsf{Pi}_{\mathsf{OB}}$, we must have $Pr_c[(x, \tau, v, \alpha, \boxed{f_{fold}}, c) \notin \mathsf{DoomedSet}_{\mathrm{HIOP}}] \leq \epsilon_{\mathsf{OB}} \leq \kappa$. Thus, this case does not occur either.

For the rest of the rounds, the probability

$$\Pr_c[(x, \tau, v, \alpha, \tau', m, c) \notin \mathsf{DoomedSet}_{\mathrm{HIOP}}]$$

when $(x, \tau, v, \alpha, \tau')$ $(\tau' \neq \emptyset)$ is clearly bounded by ϵ_{OB} because this event occurs only when $(\{\boxed{f_i}\}_{i=1}^n, \alpha, \tau') \notin \mathsf{DoomedSet}_{\mathsf{OB}}$ but $(\{\boxed{f_i}\}_{i=1}^n, \alpha, \tau', m, c) \in \mathsf{DoomedSet}_{\mathsf{OB}}$. □

5 Split accumulation and PCD

Let us recall our notations.

Input: qi **Output:** $\mathsf{pk}_{\mathsf{SA}}, \mathsf{vk}_{\mathsf{SA}}, \mathsf{dk}_{\mathsf{SA}}$	**Input:** $\mathsf{dk}_{\mathsf{SA}}, \mathsf{acc}$ **Output:** $0/1$
$(\mathsf{pk}_{\mathsf{NARK}}, \mathsf{vk}_{\mathsf{NARK}}) \leftarrow \mathsf{NARK.I}^{\mathcal{H}}(\mathsf{qi})$ $\mathbb{i}_{\mathsf{NOB}} \leftarrow \mathtt{Parse}(\mathsf{vk}_{\mathsf{NARK}})$ $(\mathsf{pk}_{\mathsf{NOB}}, \mathsf{vk}_{\mathsf{NOB}}) \leftarrow \mathsf{NOB.I}^{\mathcal{H}}(\mathbb{i}_{\mathsf{NOB}})$ $\mathsf{pk}_{\mathsf{SA}} \leftarrow \mathsf{pk}_{\mathsf{NARK}} \,\|\, \mathsf{pk}_{\mathsf{NOB}}$ $\mathsf{vk}_{\mathsf{SA}} \leftarrow \mathsf{vk}_{\mathsf{NARK}} \,\|\, \mathsf{vk}_{\mathsf{NOB}}$ $\mathsf{dk}_{\mathsf{SA}} \leftarrow \mathsf{vk}_{\mathsf{NOB}}$	$\mathsf{vk}_{\mathsf{NOB}} \leftarrow \mathtt{Parse}(\mathsf{dk}_{\mathsf{SA}})$ $\boxed{f} \leftarrow \mathtt{ParseDesc}(\mathsf{acc}.\mathbb{x})$ $v \leftarrow \mathsf{NOB.V}_2^{\mathcal{H}}(\boxed{f}, \mathsf{vk}_{\mathsf{NOB}}, \mathsf{acc}.\mathbb{w})$ **return** v

Fig. 1: *Procedures* $\mathsf{SA}.\mathbf{I}^{\mathcal{H}}$ *(on the left) and* $\mathsf{SA}.\mathbf{D}^{\mathcal{H}}$ *(on the right).*

- $\mathtt{ParseDesc}(\mathsf{vk}/\mathsf{pk}, \pi.\mathbb{x})$ parses the proof and outputs a description $\boxed{f}$ for which the proximity test is performed.
- $\mathtt{ParseEval}(\mathsf{pk}, \pi.\mathbb{w})$ and $\mathtt{ParseAP}(\mathsf{pk}, \pi.\mathbb{w})$ parse the proof and output a vector of evaluations v_i and corresponding auth paths ap_i, respectively.
- $\mathsf{CheckAP}(\boxed{f}, \mathsf{pk}, \pi.\mathbb{w})$ parses the proof and verifies authentication paths with respect to $\boxed{f}$.

Let Π be a RBR knowledge sound δ-correlated HIOP for an indexed regular (non-oracle) relation $\mathcal{R} = (\mathbb{i}, \mathbb{x}, \mathbb{w})$ and

$$\mathsf{NARK} = \mathsf{BCST}^{\perp}(\mathsf{T}[\delta, 0](\mathsf{B}(\Pi), \Pi_{\mathsf{TRV}})) = (\mathsf{I}, \mathsf{P}, \mathsf{V} = (\mathsf{V}_1, \mathsf{V}_2)).$$

This is a NARK by Lemmas 5 and 6.

In this section, we build a split accumulation scheme SA for the non-interactive argument system NARK. We will use the relation $\mathcal{R}_{\mathcal{V}}$ defined below:

$$\mathcal{R}_{\mathcal{V}} = \left\{ \begin{pmatrix} \mathsf{qi} \\ \mathsf{qx} \\ \mathsf{qw} \end{pmatrix} = \begin{pmatrix} \mathbb{i} \\ (\mathbb{x}, \pi.\mathbb{x}) \\ \pi.\mathbb{w} \end{pmatrix} \mid \begin{array}{c} (\mathsf{pk}_{\mathsf{NARK}}, \mathsf{vk}_{\mathsf{NARK}}) \leftarrow \mathsf{NARK.I}^{\mathcal{H}}(\mathbb{i}), \\ \mathsf{NARK.V}^{\mathcal{H}}(\mathsf{vk}_{\mathsf{NARK}}, \mathbb{x}, (\pi.\mathbb{x}, \pi.\mathbb{w})) = 1 \end{array} \right\}.$$

We will also define a non-interactive version of the oracle batching protocol, a key block in our design.

$$\mathsf{NOB} = \mathsf{BCST}^{\perp}(\mathsf{T}[\delta, 0](\mathsf{OB}, \Pi_{\mathsf{TRV}})).$$

The accumulator is represented by a short part $\mathsf{acc}.\mathbb{x}$, containing the description of the oracle $\boxed{f}$, and a witness part $\mathsf{acc}.\mathbb{w}$, including authentication paths to the vector of the evaluations of function f. In practice, all the authentication paths in witness part can be represented by a single hash code corresponding to the root of the Merkle-tree. The introduced notations make calls $\mathtt{ParseDesc}(\mathsf{acc}.\mathbb{x})$ and $\mathtt{ParseEval}(\mathsf{acc}.\mathbb{w})$ legitimate.

Formally, the split accumulation scheme SA is defined by the algorithms depicted in Figures 2 and 1.

Input: $\mathsf{pk}_{\mathsf{SA}}$, $(\mathsf{qx}_i, \mathsf{qw}_i)_{i=1}^n$, $(\mathsf{acc}_j)_{j=1}^m$
Output: $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$, π_{acc}

$(\mathsf{pk}_{\mathsf{NARK}}, \mathsf{pk}_{\mathsf{NOB}}) \leftarrow \mathtt{Parse}(\mathsf{pk}_{\mathsf{SA}})$
for $i \in [n]$ **do**
  $(\mathbb{x}_i, \pi_i.\mathbb{x}, \pi_i.\mathbb{w}) \leftarrow \mathtt{Parse}(\mathsf{qx}_i, \mathsf{qw}_i)$
  $\boxed{f_i} \leftarrow \mathtt{ParseDesc}(\mathsf{pk}_{\mathsf{NARK}}, \pi_i.\mathbb{x})$
  $f_i \leftarrow \mathtt{ParseEval}(\mathsf{pk}_{\mathsf{NARK}}, \pi_i.\mathbb{w})$
end for
for $j \in [n+1, n+m]$ **do**
  $\boxed{f_j} \leftarrow \mathtt{ParseDesc}(\mathsf{pk}_{\mathsf{SA}}, \mathsf{acc}_j.\mathbb{x})$
  $f_j \leftarrow \mathtt{ParseEval}(\mathsf{pk}_{\mathsf{SA}}, \mathsf{acc}_j.\mathbb{w})$
end for
$\mathbb{x}' \leftarrow \boxed{f_1, \ldots, f_{n+m}}$
$\mathbb{w}' \leftarrow (f_1, \ldots, f_{n+m})$
$\mathsf{pf}_{\mathsf{NOB}} \leftarrow \mathsf{NOB}.\mathsf{P}^{\mathcal{H}}(\mathsf{pk}_{\mathsf{NOB}}, \mathbb{x}', \mathbb{w}')$
$(\pi_{\mathsf{NOB}}.\mathbb{x}, \pi_{\mathsf{NOB}}.\mathbb{w}) \leftarrow \mathtt{Parse}(\mathsf{pf}_{\mathsf{NOB}})$
$\mathsf{acc}.\mathbb{x} \leftarrow \mathtt{ParseDesc}(\mathsf{pk}_{\mathsf{NOB}}, \pi_{\mathsf{NOB}}.\mathbb{x})$
$\mathsf{acc}.\mathbb{w} \leftarrow \pi_{\mathsf{NOB}}.\mathbb{w}$
$\pi_{\mathsf{acc}} \leftarrow \pi_{\mathsf{NOB}}.\mathbb{x}$
return $(\mathsf{acc}, \pi_{\mathsf{acc}})$

Input: $\mathsf{vk}_{\mathsf{SA}}$, $(\mathsf{qx}_i)_{i=1}^n$, $(\mathsf{acc}_j.\mathbb{x})_{j=1}^m$, π_{acc}, $\mathsf{acc}.\mathbb{x}$
Output: $0/1$

$(\mathsf{vk}_{\mathsf{NARK}}, \mathsf{vk}_{\mathsf{NOB}}) \leftarrow \mathtt{Parse}(\mathsf{vk}_{\mathsf{SA}})$
for $i \in [n]$ **do**
  $(\mathbb{x}_i, \pi_i.\mathbb{x}) \leftarrow \mathtt{Parse}(\mathsf{qx}_i)$
  $v_i \leftarrow \mathsf{NARK}.\mathsf{V}_1^{\mathcal{H}}(\mathsf{vk}_{\mathsf{NARK}}, \mathbb{x}_i, \pi_i.\mathbb{x})$
  $\boxed{f_i} \leftarrow \mathtt{ParseDesc}(\mathsf{vk}_{\mathsf{NARK}}, \pi_i.\mathbb{x})$
end for
for $j \in [n+1, n+m]$ **do**
  $\boxed{f_j} \leftarrow \mathtt{ParseDesc}(\mathsf{vk}_{\mathsf{SA}}, \mathsf{acc}_j.\mathbb{x})$
end for
$\mathbb{x}' \leftarrow \boxed{f_1, \ldots, f_{n+m}}$
$v_{n+1} \leftarrow \mathsf{NOB}.\mathsf{V}_1^{\mathcal{H}}(\mathsf{vk}_{\mathsf{NOB}}, \mathbb{x}', \pi_{\mathsf{acc}})$
$v_{n+2} \leftarrow (\mathtt{ParseDesc}(\mathsf{vk}_{\mathsf{SA}}, \mathsf{acc}.\mathbb{x}) = \mathtt{ParseDesc}(\mathsf{vk}_{\mathsf{NOB}}, \pi_{\mathsf{acc}}))$
return $\bigwedge_{i=1}^{n+2} v_i$

Fig. 2: *Procedures* $\mathsf{SA}.\mathbf{P}^{\mathcal{H}}$ *(on the left) and* $\mathsf{SA}.\mathbf{V}^{\mathcal{H}}$ *(on the right).*

Theorem 3. SA *is complete.*

Proof. We will show that the probability

$$\Pr\left[\begin{array}{c} (\mathsf{pk}_{\mathsf{SA}}, \mathsf{vk}_{\mathsf{SA}}, \mathsf{dk}_{\mathsf{SA}}) \leftarrow \mathsf{SA}.\mathbf{I}^{\mathcal{H}}(\mathsf{qi}) \\ \begin{pmatrix} \mathsf{acc}, \\ \pi_{\mathsf{acc}} \end{pmatrix} \leftarrow \mathsf{SA}.\mathbf{P}^{\mathcal{H}}\left(\begin{array}{c} \mathsf{pk}_{\mathsf{SA}}, \\ (\mathsf{qx}_i, \mathsf{qw}_i)_{i=1}^n, \\ (\mathsf{acc}_j)_{j=1}^m \end{array}\right) \end{array} : \begin{array}{c} (\mathsf{qi}, \mathsf{qx}_i, \mathsf{qw}_i) \in \mathcal{R}_{\mathcal{V}}\ \forall i \in [n] \\ 1 \leftarrow \mathsf{SA}.\mathbf{D}^{\mathcal{H}}(\mathsf{dk}_{\mathsf{SA}}, \mathsf{acc}_j)\ \forall j \in [m] \\ \Downarrow \\ 1 \leftarrow \mathsf{SA}.\mathbf{V}^{\mathcal{H}}\left(\begin{array}{c} \mathsf{vk}_{\mathsf{SA}}, (\mathsf{qx}_i)_{i=1}^n, \\ (\mathsf{acc}_j.\mathbb{x})_{j=1}^m, \\ \mathsf{acc}.\mathbb{x}, \pi_{\mathsf{acc}} \end{array}\right) \\ 1 \leftarrow \mathsf{SA}.\mathbf{D}^{\mathcal{H}}(\mathsf{dk}_{\mathsf{SA}}, \mathsf{acc}) \end{array}\right]$$

is equal to 1. $\forall i \in [n], \forall j \in [m]$

$$\begin{cases} (\mathsf{qi}, \mathsf{qx}_i, \mathsf{qw}_i) \in \mathcal{R}_{\mathcal{V}}, \\ \mathsf{SA}.\mathbf{D}^{\mathcal{H}}(\mathsf{dk}_{\mathsf{SA}}, \mathsf{acc}_j) = 1 \end{cases} \Rightarrow \begin{cases} \mathsf{NARK}.\mathsf{V}_2^{\mathcal{H}}(\boxed{f_i}, \mathsf{vk}_{\mathsf{NARK}}, \pi_i.\mathbb{w}) = 1, \\ \mathsf{NOB}.\mathsf{V}_2^{\mathcal{H}}(\boxed{f_{n+j}}, \mathsf{vk}_{\mathsf{NOB}}, \mathsf{acc}_j.\mathbb{w}) = 1, \end{cases}$$

where

– $(\mathsf{vk}_{\mathsf{NARK}}, \mathsf{vk}_{\mathsf{NOB}}) \leftarrow \mathtt{Parse}(\mathsf{vk}_{\mathsf{SA}})$

– $(\mathsf{pk}_{\mathsf{NARK}}, \mathsf{pk}_{\mathsf{NOB}}) \leftarrow \mathtt{Parse}(\mathsf{pk}_{\mathsf{SA}})$
– $((\mathbb{x}_i, \pi_i.\mathbb{x}), \pi_i.\mathbb{w}) \leftarrow \mathtt{Parse}(\mathsf{qx}_i, \mathsf{qw}_i)\ \forall i \in [n]$
– $\boxed{f_i} \leftarrow \mathtt{ParseDesc}(\mathsf{vk}_{\mathsf{NARK}}, \pi_i.\mathbb{x})\ \forall i \in [n]$
– $f_i \leftarrow \mathtt{ParseEval}(\mathsf{pk}_{\mathsf{NARK}}, \pi_i.\mathbb{w})\ \forall i \in [n]$
– $\boxed{f_{n+j}} \leftarrow \mathtt{ParseDesc}(\mathsf{pk}_{\mathsf{SA}}, \mathsf{acc}_j.\mathbb{x})\ \forall j \in [m]$
– $f_{n+j} \leftarrow \mathtt{ParseEval}(\mathsf{pk}_{\mathsf{SA}}, \mathsf{acc}_j.\mathbb{w})\ \forall j \in [m]$

In particular, this means that

$$(\mathbb{i}_{\mathsf{NOB}}, \boxed{\boldsymbol{f_i}}, \boldsymbol{f_i}) \in \mathsf{CoAgg}(0),$$

where $\mathbb{i}_{\mathsf{NOB}} \leftarrow \mathtt{Parse}(\mathsf{vk}_{\mathsf{NARK}})$. And with probability 1,

$$\mathsf{NOB}.\mathsf{V}_2^{\mathcal{H}}(\boxed{g}, \mathsf{vk}_{\mathsf{NOB}}, \pi_{\mathsf{NOB}}.\mathbb{w}) = 1, \text{ where}$$

– $\mathbb{x}' = \boxed{f_1, \ldots, f_{n+m}}$,
– $\mathbb{w}' = (f_1, \ldots, f_{n+m})$,
– $\mathsf{NOB}.\mathsf{P}^{\mathcal{H}}(\mathsf{pk}_{\mathsf{NOB}}, \mathbb{x}', \mathbb{w}') \to (\pi_{\mathsf{NOB}}.\mathbb{x}, \pi_{\mathsf{NOB}}.\mathbb{w})$,
– $\boxed{g} \leftarrow \mathtt{ParseDesc}(\mathsf{pk}_{\mathsf{NOB}}, \pi_{\mathsf{NOB}}.\mathbb{x}) = \mathtt{ParseDesc}(\mathsf{acc}.\mathbb{x})$.

In other words, $\mathsf{SA}.\mathbf{D}^{\mathcal{H}}(\mathsf{dk}_{\mathsf{SA}}, \mathsf{acc}) = 1$. We have already shown that

$$\begin{cases} \forall i \in [n]\ \mathsf{NARK}.\mathsf{V}_1^{\mathcal{H}}(\mathsf{vk}_{\mathsf{NARK}}, \mathbb{x}_i, \pi_i.\mathbb{x}) = 1, \\ \mathsf{NOB}.\mathsf{V}_1^{\mathcal{H}}(\mathsf{vk}_{\mathsf{NOB}}, \mathbb{x}', \pi_{\mathsf{NOB}}.\mathbb{x}) = 1. \end{cases}$$

This means that

$$\mathsf{SA}.\mathbf{V}^{\mathcal{H}}\left(\mathsf{vk}_{\mathsf{SA}}, (\mathsf{qx}_i)_{i=1}^n, (\mathsf{acc}_j.\mathbb{x})_{j=1}^m, \mathsf{acc}.\mathbb{x}, \pi_{\mathsf{acc}}\right) = 1 \text{ with probability 1.}$$

□

Following [BMNW24a], it is enough to show the knowledge soundness relative to the relaxed verifier and decider. In our terms, this means considering specified property relative to the following systems

$$\mathsf{NARK}^{\perp} = \mathsf{BCST}^{\perp}(\mathsf{T}[\delta, \delta](\mathsf{B}(\Pi), \Pi_{\mathsf{TRV}})),$$
$$\mathsf{NOB}^{\perp} = \mathsf{BCST}^{\perp}(\mathsf{T}[\delta, \delta](\mathsf{OB}, \Pi_{\mathsf{TRV}})).$$

Theorem 4. SA *is knowledge sound with respect to* $\mathsf{NARK}^{\perp}$ *and* $\mathsf{NOB}^{\perp}$.

Lemma 7. *Let* $\delta_0 \leq \delta$, *and let* $\Pi = \mathsf{T}[\delta, \delta_0](\mathsf{OB}, \Pi_{\mathsf{TRV}})$ *be a HIOP for oracle relation* $\mathsf{CoAgg}(\delta)$. *Let* $\mathsf{NOB}^{\perp} = \mathsf{BCST}^{\perp}(\Pi)$. *Then there exists a PPT extractor* $\mathsf{E}_{\mathsf{NOB}^{\perp}}$ *such that, for any PPT* $\tilde{P}$ *with random oracle query bound* Q, *and for any* $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$, *we have that*

$$\Pr\left[\begin{array}{c} \mathsf{NOB}^{\perp}.\mathsf{V}^{\mathcal{H}}(\mathsf{vk}_{\mathsf{NOB}}, \mathbb{x}, \pi) = 1 \\ \wedge \\ \left(\begin{array}{c} (\mathbb{i}_{\mathsf{NOB}}, \boxed{(g_i)_{i=1}^n}, (g_i)_{i=1}^n) \notin \mathsf{CoAgg}(\delta) \\ \vee \\ \exists i \in [n] : \mathtt{CheckAP}(\boxed{f_i}, \mathsf{vk}_{\mathsf{NOB}}, \hat{\pi}_i) = 0 \end{array}\right) \end{array} : \begin{array}{c} (\mathbb{i}_{\mathsf{NOB}}, \mathsf{vk}_{\mathsf{NOB}}, \mathsf{pk}_{\mathsf{NOB}}, \mathbb{x}, \pi) \leftarrow \tilde{P}^{\mathcal{H}} \\ (\hat{\pi}_i)_{i=1}^n \leftarrow \mathsf{E}_{\mathsf{NOB}^{\perp}}^{\tilde{P}, \mathcal{H}} \\ g_i \leftarrow \mathtt{ParseEval}(\mathsf{pk}_{\mathsf{NOB}}, \hat{\pi}_i) \end{array}\right]$$

is at most $\kappa_{\mathsf{NOB}^\perp}$*, where* $\kappa_{\mathsf{NOB}^\perp} = Q \cdot \varepsilon_{\mathsf{rbr}} + 3(Q^2+1)/2^k$*,* n *– number of functions in* $\mathsf{CoAgg}(\delta)$*,* $\varepsilon_{\mathsf{rbr}}$ *– round-by-round soundness error of* Π_{OB}*.*

Proof. Let us recall that for CoAgg relation $x = \boxed{w}$, and OB's verifier gets access to oracles in the first round of interaction. It is not difficult to see that $\Pi = \mathsf{T}[\delta, \delta_0](\mathsf{OB}, \Pi_{\mathsf{TRV}})$ is RBR knowledge sound. Moreover, round-by-round knowledge error $\kappa^{\Pi}_{\mathsf{rbr}}$ is equal to Π_{OB}'s round-by-round soundness error $\varepsilon_{\mathsf{rbr}}$.

Following the reasoning of Lemma 5, we can construct an extractor $\tilde{E}$, which allows us, except with probability at most $Q \cdot \varepsilon^{\mathsf{OB}}_{\mathsf{rbr}} + 3(Q^2+1)/2^k$, to obtain $(g_i)_{i=1}^n$ such that $(\mathbb{i}_{\mathsf{NOB}}, \boxed{(g_i)_{i=1}^n}, (g_i)_{i=1}^n) \in \mathsf{CoAgg}$.

It remains for us to show how we can obtain vectors of authentication paths $(\hat{\pi}_i)_i$ consistent with $(\boxed{f_i})_i$. Recall that the specified extractor replaces unknown elements of the partial openings of the Merkle-tree commitment with zeros. For each such replaced element, we assign an authentication path $\perp$. For the remaining elements, having full table of random oracle calls, we can restore the authentication path consistent with the original commitments $(\boxed{f_i})_i$. □

Proof. Now we move on to proving Theorem 4. We need to show that the following probability:

$$\Pr\left[\begin{array}{c} \mathsf{ai} \leftarrow \mathcal{D}(1^\lambda) \\ \begin{pmatrix} \mathsf{qi}, (\mathsf{qx}_i)_{i=1}^n, \\ (\mathsf{acc}_j.\mathbb{x})_{j=1}^m, \\ \mathsf{acc}, \pi_{\mathsf{acc}} \end{pmatrix} \leftarrow \tilde{\mathcal{P}}^{\mathcal{H}}(\mathsf{ai}) \\ \begin{pmatrix} (\mathsf{qw}_i)_{i=1}^n, \\ (\mathsf{acc}_j.\mathbb{w})_{j=1}^m \end{pmatrix} \leftarrow \mathsf{E}^{\tilde{P},\mathcal{H}}(\mathsf{ai}) \\ (\mathsf{pk}_{\mathsf{SA}}, \mathsf{vk}_{\mathsf{SA}}, \mathsf{dk}_{\mathsf{SA}}) \leftarrow \mathsf{SA}.\mathbf{I}^{\mathcal{H}}(\mathsf{qi}) \end{array} : \begin{array}{c} 1 \leftarrow \mathsf{SA}.\mathbf{V}^{\mathcal{H}}\begin{pmatrix} \mathsf{vk}_{\mathsf{SA}}, (\mathsf{qx}_i)_{i=1}^n, \\ (\mathsf{acc}_j.\mathbb{x})_{j=1}^m, \\ \mathsf{acc}.\mathbb{x}, \pi_{\mathsf{acc}} \end{pmatrix} \\ 1 \leftarrow \mathsf{SA}.\mathbf{D}^{\mathcal{H}}(\mathsf{dk}_{\mathsf{SA}}, \mathsf{acc}) \\ \Downarrow \\ (\mathsf{qi}, \mathsf{qx}_i, \mathsf{qw}_i) \in \mathcal{R}_{\mathcal{V}}\ \forall i \in [n] \\ 1 \leftarrow \mathsf{SA}.\mathbf{D}^{\mathcal{H}}(\mathsf{dk}, \mathsf{acc}_j)\ \forall j \in [m] \end{array}\right]$$

is negligibly close to 1. It is easy to see that

$$\mathsf{NOB}^\perp.\mathsf{V}^{\mathcal{H}}(\mathsf{vk}_{\mathsf{NOB}}, \mathbb{x}', (\pi_{\mathsf{acc}}, \mathsf{acc}.\mathbb{w})) = 1, \text{ where}$$

- $(\mathsf{vk}_{\mathsf{NARK}}, \mathsf{vk}_{\mathsf{NOB}}) \leftarrow \mathtt{Parse}(\mathsf{vk}_{\mathsf{SA}})$,
- $(\mathsf{pk}_{\mathsf{NARK}}, \mathsf{pk}_{\mathsf{NOB}}) \leftarrow \mathtt{Parse}(\mathsf{pk}_{\mathsf{SA}})$,
- $(\mathbb{x}_i, \pi_i.\mathbb{x}) \leftarrow \mathtt{Parse}(\mathsf{qx}_i) \forall i \in [n]$,
- $\boxed{f_i} \leftarrow \mathtt{ParseDesc}(\mathsf{vk}_{\mathsf{NARK}}, \pi_i.\mathbb{x}) \forall i \in [n]$,
- $\boxed{f_{n+j}} \leftarrow \mathtt{ParseDesc}(\mathsf{pk}_{\mathsf{SA}}, \mathsf{acc}_j.\mathbb{x}) \forall j \in [m]$,
- $\mathbb{x}' = \boxed{f_1, \ldots, f_{n+m}}$.

Following Lemma 7, except with probability at most $\kappa_{\mathsf{NOB}^\perp}$, we have that

$$\mathsf{E}^{\tilde{P},\mathcal{H}}_{\mathsf{NOB}^\perp} \to \hat{\pi}_1, \ldots, \hat{\pi}_{n+m},$$

where $\mathsf{E}_{\mathsf{NOB}^\perp}$ is a extractor for $\mathsf{NOB}^\perp$, and

$$\mathtt{CheckAP}(\boxed{f_i}, \mathsf{pk}_{\mathsf{NOB}}, \hat{\pi}_i) = 1\ \forall i \in [n+m].$$

This means that

$$\mathsf{SA}.\mathbf{D}^{\mathcal{H}}(\mathsf{dk}_{\mathsf{SA}}, (\mathsf{acc}_j.\mathbb{x}, \hat{\pi}_j)) = 1 \text{ for } j = i+1, \ldots, n+m,$$

$$\mathsf{NARK}^{\perp}.\mathsf{V}^{\mathcal{H}}(\mathsf{vk}_{\mathsf{NARK}}, \mathbb{x}_i, (\pi_i.\mathbb{x}, \hat{\pi}_i)) = 1, \ \forall i \in [n].$$

Thus, the knowledge soundness error is equal to $\kappa_{\mathsf{NOB}^{\perp}}$. □

Given NARK and the split accumulation scheme SA, we can use [BCL+21, Theorem 5.3] to build a PCD scheme. Recall that PCD allows us to show the correctness of a distributed computation. More precisely, given a compliance predicate ϕ, PCD enables untrusted provers to demonstrate that if a computational node uses local input data z_{loc}, received messages $z_1, \ldots, z_t$, and outputs z_{out}, then

$$\phi(z_{\mathsf{out}}, z_{\mathsf{loc}}, z_1, \ldots, z_t) = 1.$$

Informally, the PCD prover takes as input $z_{\mathsf{loc}}, z_{\mathsf{out}}$ and messages $(z_i)_{i=1}^t$ augmented with corresponding PCD proofs $(\pi_i^{\mathsf{PCD}})_{i=1}^t$, where

$$\pi_i^{\mathsf{PCD}} = ((\pi_i.\mathbb{x}, \pi_i.\mathbb{w}), (\mathsf{acc}_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{w})).$$

It accumulates $((z_i, \pi_i.\mathbb{x}), \pi_i.\mathbb{w}, (\mathsf{acc}_i.\mathbb{x}, \mathsf{acc}_i.\mathbb{w}))_{i=1}^t$ to obtain a new accumulator acc and accumulation proof π_{acc}. Finally, the prover uses NARK to generate a proof π_{NARK} for the following statement (presented as a circuit):

1. $\phi(z_{\mathsf{out}}, z_{\mathsf{loc}}, (z_i)_{i=1}^t) = 1$,
2. $\mathsf{SA}.\mathsf{V}((z_i, \pi_i.\mathbb{x})_{i=1}^t, (\mathsf{acc}_i.\mathbb{x})_{i=1}^t, \mathsf{acc}.\mathbb{x}, \pi_{\mathsf{acc}}) = 1$.

The PCD prover outputs the new proof $\pi^{\mathsf{PCD}} = (\pi_{\mathsf{NARK}}, \mathsf{acc})$.

6 Experimental Evaluation and Discussion

This section discusses possible optimizations of the BOIL protocol and presents an experimental evaluation.

Metrics and methodology. We implemented[5] the components of the BOIL protocol using the Plonky2 proof system. This choice was motivated by several factors. Plonky2 is highly optimized for recursive proof composition, and despite recent progress in folding schemes such as Nova [KST22], it remains a practical preference in many applications [DD23,PSG+24]. Importantly, Plonky2 uses a FRI-based commitment scheme and provides a ready-made verifier circuit, unlike Plonky3, which simplifies integration with our protocol.

We conducted experiments using our implementation on a consumer-grade laptop (Apple M2 MacBook Pro). For the baseline, we used Plonky2 with the following parameters: Reed–Solomon codes with a rate of 1/8, the number of proof-of-work bits set to 0, and a FRI reduction strategy of $(4, 5)$. This reduction strategy means that at the i-th iteration of the FRI commit phase, the domain size is reduced by a factor of 2^4. Once the polynomial's degree falls to 2^5 or less, it is sent directly to the verifier.

[5] https://github.com/smilesmirk/Boil

Next, we replaced the FRI with the OB protocol and measured the relevant performance metrics for various values of the polynomial degree d.

Selection of security parameters. The results and theoretical estimates presented in Chapter 5 provide an explicit method for selecting parameters of OB protocol that ensure a proven level of completeness and soundness for the resulting PCD scheme. Following the approach used in Plonky2 and other practical implementations, we selected parameters under the assumption of the Reed–Solomon decoding conjecture [BCI+20, Conjecture 8.4]. To target 128-bit security, we choose the number of sampled points t such that $t \cdot \log(1/\rho) \geq 128$. For our configuration, we set $s = 2$ and $t = 43$.

Degree homogenization. In practice, it is useful to think of the accumulator as a vector of $s + t$ polynomials of degree $d - 1$: $f^{(i)}_{\mathsf{new}} = \mathsf{Quotient}(f_{\mathsf{fold}}, \boldsymbol{x}_i, \boldsymbol{y}_i)$, where $\boldsymbol{x} = (x_1^{\mathsf{out}}, \ldots, x_s^{\mathsf{out}}, x_1, \ldots, x_t)$ and $\boldsymbol{y} = (y_1^{\mathsf{out}}, \ldots, y_s^{\mathsf{out}}, y_1, \ldots, y_t)$. Note that we can compute all $s + t$ evaluations of $\boxed{f^{(i)}_{\mathsf{new}}}$ at a given point g using just a single query to $\boxed{f_{\mathsf{fold}}}$. As a result, we can still efficiently verify that the linear combination $\sum_{i=1}^{s+t} \beta^i \cdot f^{(i)}_{\mathsf{new}}$ is close to a Reed–Solomon code for a randomly chosen β, while avoiding the need to interpolate the values in $\boldsymbol{y}$. This reduces both the verification complexity and the size of the corresponding circuit.

Furthermore, recall that the final step of HIOP involves checking whether the set of maps $\mathsf{Quotient}(g_i, \zeta_i, g_i(\zeta_i))$ has a δ-correlated agreement. Since all input and output polynomials of OB protocol have degree $d - 1$, the use of DegCor is unnecessary in this setting.

Table 1: *Prover's performance and proof sizes*

	Proving Time, sec			Size, MB	
	FRI	OB	Total	Witness	Accumulator
$\log(d) = 18$	0.49	0.68	≈ 16	270	4
$\log(d) = 19$	1.38	1.51	≈ 150	540	8
$\log(d) = 20$	3.87	3.83	≈ 2100	1080	16

Proving time and state size. The execution times of the OB protocol and FRI are comparable (see Table 1), with the difference being negligible due to their minor contribution to the overall proving time. The vast majority of the remaining time is spent committing to all witness elements. Therefore, it is more informative to assess the size of the recursive circuit, which represents the per-step overhead in the recursive proving process.

By using the OB protocol both in the accumulator scheme and in the NARK, we obtain a significantly smaller PCD proof size. Specifically, for a Plonkish circuit represented by a $d \times M$ matrix (with $M = 135$ in Plonky2), the proof size becomes proportional to the size of a single column – i.e., d elements. In practice,

this leads to a reduction in proof size by several orders of magnitude. This compression enables more efficient parallelization of computation. In contrast, folding-based schemes generally require storing the entire witness as part of the state.

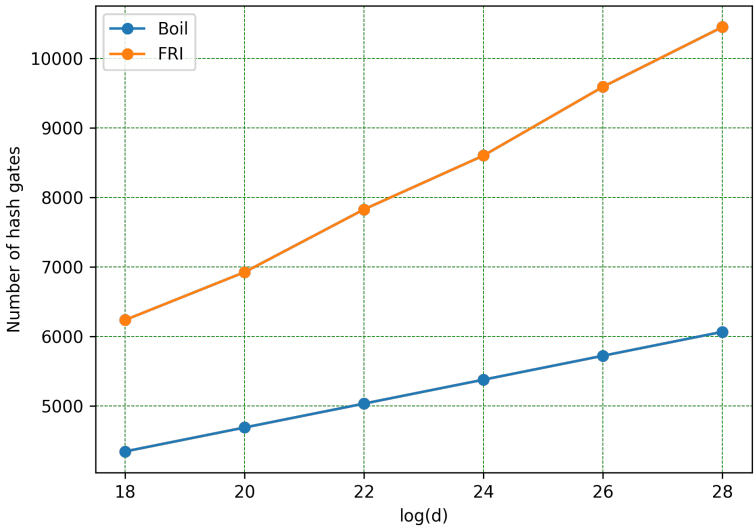

Fig. 3: *Size of the verification circuit*

Recursive Circuit Size. Finally, we estimated the size of the verification circuit in terms of the number of hash gates. As shown in Figure 3, the use of the BOIL protocol yields a noticeable reduction in circuit size for practically relevant parameter settings.

References

ACFY24a. Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: Reed-solomon proximity testing with fewer queries. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 380–413. Springer, Cham, August 2024. 3, 5, 8, 9, 14

ACFY24b. Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. Whir: Reed–solomon proximity testing with super-fast verification. *Cryptology ePrint Archive*, 2024. 14

BBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018. 2, 4, 14

BC23. Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 77–110. Springer, Singapore, December 2023. 2, 6

BCCT12. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. Cryptology ePrint Archive, Report 2012/095, 2012. 1

BCI+20. Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed-solomon codes. In *61st FOCS*, pages 900–909. IEEE Computer Society Press, November 2020. 2, 5, 8, 31

BCL+21. Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 681–710, Virtual Event, August 2021. Springer, Cham. 1, 4, 14, 15, 30

BCMS20. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Report 2020/499, 2020. 1

BCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Berlin, Heidelberg, October / November 2016. 6, 11, 22, 23

BCTV14. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Berlin, Heidelberg, August 2014. 1

BDFG21. Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 649–680, Virtual Event, August 2021. Springer, Cham. 1, 3

BGH19. Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. 1

BGK+23. Alexander R. Block, Albert Garreta, Jonathan Katz, Justin Thaler, Pratyush Ranjan Tiwari, and Michal Zajac. Fiat-shamir security of FRI and related SNARKs. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 3–40. Springer, Singapore, December 2023. 3, 5, 6, 7, 10, 11, 12, 13, 24

BMNW24a. Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang. Accumulation without homomorphism. Cryptology ePrint Archive, Report 2024/474, 2024. 2, 3, 4, 5, 6, 7, 28

BMNW24b. Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang. Arc: Accumulation for reed–solomon codes, 2024. Publication info: Preprint. 7

CBBZ23. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 499–530. Springer, Cham, April 2023. 3

Chi10. Alessandro Chiesa. Proof-carrying data, 2010. Accepted: 2011-02-23T14:20:59Z Journal Abbreviation: PCD. 1

CMS19. Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 1–29. Springer, Cham, December 2019. 11

COS20. Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Cham, May 2020. 1, 2, 3, 7, 10, 11

CY24. Alessandro Chiesa and Eylon Yogev. Building cryptographic proofs from hash functions, 2024. 9

DD23. Sai Deng and Bo Du. zkTree: a zk recursion tree with ZKP membership proofs. Cryptology ePrint Archive, Report 2023/208, 2023. 30

EG23. Liam Eagen and Ariel Gabizon. ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances. Cryptology ePrint Archive, Report 2023/1106, 2023. 2, 6

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. 11

GKR+21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021. 3

GW20a. Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020. 6

GW20b. Ariel Gabizon and Zachary J. Williamson. Proposal: The turbo-plonk program syntax for specifying snark programs, 2020. 6

GWC19a. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. 4

GWC19b. Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019. 12

Hab22. Ulrich Haböck. A summary on the FRI low degree test. Cryptology ePrint Archive, Report 2022/1216, 2022. 2

KNS24. Tohru Kohrita, Maksim Nikolaev, and Javier Silva. Distributed proof generation for zkEVM from code-based polynomial commitment schemes. Talk at ZK Summit 12, Lisbon, Portugal, October 8th, 2024. 7

KPV22. Assimakis A. Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitments. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1725–1737. ACM Press, November 2022. 2, 4, 5

KS22. Abhiram Kothapalli and Srinath Setty. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758, 2022. 6

KS23. Abhiram Kothapalli and Srinath Setty. CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Report 2023/1192, 2023. 6

KS24. Abhiram Kothapalli and Srinath T. V. Setty. HyperNova: Recursive arguments for customizable constraint systems. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 345–379. Springer, Cham, August 2024. 6

KST22. Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 359–388. Springer, Cham, August 2022. 2, 6, 30

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010. 2

Mic00. Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000. 11

NBS23. Wilson Nguyen, Dan Boneh, and Srinath Setty. Revisiting the nova proof system on a cycle of curves. Cryptology ePrint Archive, Report 2023/969, 2023. 6

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Berlin, Heidelberg, August 1992. 2

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Berlin, Heidelberg, May 1996. 11

PSG$^+$24. Charalampos Papamanthou, Shravan Srinivasan, Nicolas Gailly, Ismael Hishon-Rezaizadeh, Andrus Salumets, and Stjepan Golemac. Reckle trees: Updatable merkle batch proofs with applications. Cryptology ePrint Archive, Report 2024/493, 2024. 30

Sou23. Lev Soukhanov. Reverie: an end-to-end accumulation scheme from cyclefold. Cryptology ePrint Archive, Report 2023/1888, 2023. 7

Sta21. StarkWare. ethSTARK documentation. Cryptology ePrint Archive, Report 2021/582, 2021. 2

Sze24. Alan Szepieniec. DEEP commitments and their applications, 2024. Publication info: Preprint. 7

Teaa. Polygon Zero Team. Plonky2: Fast recursive arguments with plonk and fri. 2, 4

Teab. RISC Zero Team. Zero-knowledge verifiable general computing platform based on zk-starks and the risc-v microarchitecture. 2, 4

Val08. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, March 2008. 1, 11