# Vector Commit-and-Reduce Schemes

Anubhav Baweja
abaweja@upenn.edu
University of Pennsylvania

Pratyush Mishra
prat@upenn.edu
University of Pennsylvania

Tushar Mopuri
tmopuri@upenn.edu
University of Pennsylvania

Bharath Namboothiry
namboo@upenn.edu
University of Pennsylvania

November 5, 2026

**Abstract**

Hash-based argument systems are seeing increasing adoption in practice, owing to their ease of deployment, fast provers, and plausible post-quantum security. However, they still lag behind elliptic-curve counterparts due to their large proof sizes and (comparatively more) expensive verifiers. Attempts to optimize these metrics have focused on one of the two ingredients underlying these arguments, namely Interactive Oracle Reductions, leaving the other ingredient, Vector Commitments (VC) schemes largely ignored. This is in part because Merkle trees are the best known hash-based VC scheme to date.

To make progress on this front, we introduce Vector Commit-and-Reduce (VCR) schemes. These generalize VC schemes by allowing both the commitment and opening algorithms to be interactive protocols, and moreover by allowing the opening algorithm to be a *reduction* instead of an argument. By requiring VCR schemes to satisfy strong round-by-round knowledge soundness notions, we show how they can be used to construct hash-based reductions from IORs. Our transformation extends prior work which used (non-interactive) VC schemes.

We also provide a construction of a VCR scheme, Canopy, that greatly reduces proof size and verifier time by reducing the vector opening claims to proximity claims about certain (committed) codewords. This latter feature enables us to use a wide variety of techniques for proving, batching, and accumulating these claims, and indeed we use these to obtain both concrete and asymptotic improvements for hash-based arguments and reductions.

For SNARKs based on the Bolt code [CRYPTO 2026], we reduce proof sizes by 50% and verifier hashes by $3\times$ with only a 5-15% prover overhead. For SNARKs based on the Reed–Solomon code, we reduce proof sizes by 20%, and verifier hashes by $1.5\times$ with only a 20% prover overhead. We also construct hash-based Proof-Carrying Data where the recursion overhead is $O(\lambda\sqrt{\log N})$ hashes; in comparison, the state-of-the-art requires $\Omega(\lambda \log N)$ hashes.

# Contents


**1 Introduction** **3**
- 1.1 Contributions . . . 4
- 1.2 Related work . . . 7

**2 Techniques** **8**
- 2.1 Knowledge soundness of VCR schemes . . . 8
- 2.2 From IORs to IRs via VCR schemes . . . 9
- 2.3 Canopy: a VCR scheme based on error-correcting codes . . . 10
- 2.4 Constructing SNARKs and accumulation schemes from VCR schemes . . . 13

**3 Preliminaries** **14**
- 3.1 Relations . . . 14
- 3.2 Algebraic background . . . 15
- 3.3 Interactive oracle reductions (IORs) . . . 16
- 3.4 Interactive reductions in the ROM . . . 18

**4 Vector Commit-and-Reduce (VCR) schemes** **20**
- 4.1 Formal definition . . . 20
- 4.2 Vector commitment schemes as VCR schemes . . . 23
- 4.3 Merkle trees . . . 25

**5 Transforming IORs into IRs with VCR schemes** **28**
- 5.1 Committed and bad certificate relations . . . 28
- 5.2 Parameters and notation . . . 29
- 5.3 The transformation construction . . . 30
- 5.4 Eliminating the bad certificate relation . . . 33

**6 Canopy: a VCR scheme from high-distance codes** **36**

**7 Transforming IORs into SNARKs with VCR schemes** **41**
- 7.1 The generic transformation . . . 41
- 7.2 An optimized transformation using Canopy . . . 42

**8 Constructing hash-based accumulation schemes using Canopy** **44**
- 8.1 Accumulation schemes in the random oracle model . . . 44
- 8.2 Interactive accumulation schemes for $\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}}$ . . . 45
- 8.3 Canopy-based accumulation scheme construction . . . 47

**9 Implementation and evaluation** **54**
- 9.1 Baselines and experiments . . . 54
- 9.2 Parameter choices . . . 55
- 9.3 Experimental results . . . 56

**AI use disclosure** **60**

**References** **60**

**A Knowledge soundness proof of IOR-to-IR transformation** **64**

**B Proofs for Canopy** **71**

**C Properties of RBRKS interactive reductions** **75**

**D Proving a batch of SEP claims** **81**

# 1 Introduction

Succinct cryptographic argument systems are powerful cryptographic tools that have been used to improve the security, efficiency, and privacy of a wide range of applications, including decentralized systems [Mina; KB20; BMRS20; CCDW20; BCG24], provenance claims about edited media [NT16; DCB25; ZYO+25; FGZ+26], verifiable machine learning [LXZ21; WYX+21; GGJ+23], and auditable public key infrastructure [TFZ+22; TKPS22; HSBB26; HSB+26].

In recent years, hash-based succinct arguments have become the default choice for deployment [Dra25; Fs26]. This is because they rely only on ideal hash functions or *random oracles* [Kil92; BR93; Mic00], which enables them to provide several desirable properties, including plausible post-quantum security, transparent setup, relatively good prover times, and avoidance of public-key cryptography. Unfortunately, these arguments have one key efficiency shortcoming that has limited deployment: verifier cost.

In more detail, hash-based arguments tend to incur higher verifier costs in terms of both the number of bits the verifier reads (i.e., proof size), and the running time of the verifier, measured in the number of hashes it performs. Concretely, state-of-the-art hash-based SNARKs [ACFY25] have proof sizes on the order of hundreds of kilobytes, while elliptic-curve-based counterparts can have proofs as tiny as 128 bytes [Eag25; DMS26]. Similarly, to check a claim that a circuit of size say, $2^{20}$ is satisfied, verifiers for state-of-the-art hash-based SNARKs [ACFY25] must perform several thousand hashes, which can be problem in certain contexts. For example, consider incrementally verifiable computation (IVC) based on *recursive composition* of hash-based arguments. Here the verifier is expressed as a circuit that must be proven. Depending on the hash function, this circuit translation can have 200,000 gates for state-of-the-art hash-based IVC schemes [BMNW25; BCFW25; BMMS26a], while the corresponding size for curve-based IVC schemes is only 20,000 gates. On the other hand, as mentioned earlier, curve-based arguments have slow provers and are insecure against quantum attacks. Thus, we would ideally like to get the best of both worlds by retaining the benefits of hash-based SNARKs while decreasing their verifier cost.

**Why do hash-based arguments have expensive verifiers?** Hash-based arguments are constructed from the following ingredients:

1. Interactive Oracle Reductions (IORs) [BCG+19; BMNW25], which are protocols where a prover and a verifier interact to reduce an input claim to a different, ideally simpler, claim.[1] The verifier does not have to read the prover's messages in their entirety, but can instead treat them as vector oracles and only query them at indices of its choice.
2. Vector commitment (VC) schemes, which enable one to commit to a vector and later convince a verifier that the committed vector opens to certain claimed values at the verifier's query set.

These ingredients are combined to obtain a hash-based argument as follows: (i) transform the IOR into an interactive reduction (IR) by replacing the prover's oracles with vector commitments to them and authenticating the responses to the verifier's oracle queries using VC opening proofs, and then (ii) transform the latter IR into a non-interactive reduction via the Fiat–Shamir transformation [FS86]. Prior work [BCS16; BMNW25] provides a formal description of this methodology by instantiating the VC scheme with Merkle trees, as the latter are the only non-trivial VC scheme known from ideal hashes.

So in this setting, what factors affect the verifier cost of hash-based arguments? From the IOR side, the key efficiency metric is the *query complexity* $q$, which is the number of queries the verifier makes to the prover's oracles. From the VC side, the key efficiency metric is the cost (in proof size and verifier time) of the opening proofs. When using Merkle trees as the VC scheme for a proof of a statement of size $N$, the proof size of hash-based arguments is $\Omega(q \log(N/q))$ hashes, and checking it requires computing $\Omega(q \log(N/q))$ hashes.

[1] IORs generalize Interactive Oracle Proofs (IOPs) [BCS16; RRR16]; the latter reduce the input claim to a single decision bit.

**How to reduce verifier costs?** A natural route to reducing these costs is to improve the query complexity $q$ of the underlying IOR. While this approach has been pursued extensively and with considerable success (see, e.g., [ACFY24; BCF+25; NST25; BMNW25; MZ25; BMMS26a; BMMS26b; BFRW26]), progress on this front appears to have stalled. Indeed, when targeting negligible soundness error $2^{-\lambda}$, the construction paradigm underlying all efficient IORs/IOPs requires $q = \Omega(\lambda)$ [BMMS26b]. Hence progress in reducing verifier costs in hash-based arguments seems to necessitate improving the *other* ingredient, namely VC schemes. This is a daunting task, because to date the only construction of hash-based VC schemes is the venerable Merkle tree which must incur $\log N$ hashes per query. This motivates the study of alternate strategies for proving committed vector opening claims, and this is the focus of this paper.

## 1.1 Contributions

In this work, we present a new mechanism for proving vector opening queries in hash-based arguments. We introduce *Vector Commit-and-Reduce (VCR) schemes*, a new primitive that generalizes VC schemes. We then show how to compile IORs to IRs with VCRs via a new transformation that generalizes prior ones [BCS16; BMNW25]. Finally, we construct a hash-based VCR scheme called *Canopy*, and show that using this scheme with our transformation leads to argument proof size and verifier time savings compared to the existing approach that uses Merkle trees. We discuss our contributions in more detail below.

**Vector Commit-and-Reduce (VCR) schemes** generalize standard vector commitment schemes in two ways. First, in VCR schemes, both the commitment and opening protocols are allowed to be interactive. In contrast, in most VC schemes, both protocols are usually non-interactive. Indeed, this is the case for known hash-based VC schemes, i.e. Merkle trees. Second, in VCR schemes, the opening protocol does not need to be an argument, and can instead be an interactive *reduction*. That is, instead of interacting to *decide* the opening claim(s), the prover and verifier can instead interact to *reduce* the claim(s) to a different and simpler claim. We formalize this notion via the following definition that is inspired by the prior notions of functional interactive oracle commitments [BFRW26].

**Definition 1.1** (informal, Definition 4.1)**.** A *vector commit-and-reduce* scheme $\mathsf{VCR} = (\mathsf{Com}, \mathsf{Rdx}, \mathcal{R}_{\mathsf{out}})$ consists of two protocols:

- $\mathsf{VCR.Com}$, a public-coin interactive commitment protocol that binds a prover to a single message $\boldsymbol{m}$ via an interaction with the verifier, with the interaction transcript serving as the commitment.
- $\mathsf{VCR.Rdx}$, a public-coin interactive reduction [KP23] that allows a prover to reduce claims of the form $\boldsymbol{m}[i] = v$ to claims about $\mathcal{R}_{\mathsf{out}}$.

We require VCR schemes to satisfy strong "round-by-round" knowledge soundness notions [BMNW25; BCFW25] as these will enable them to be used to for constructing hash-based non-interactive reductions of knowledge (NIRKs). We describe this construction next.

**Hash-based reductions from VCR schemes.** Like prior approaches [BCS16; BMNW25], our construction proceeds via a two-step process. First, we transform interactive oracle reductions (IORs) to interactive reductions (IRs). Our transformation proceeds round by round. For each IOR prover message $M_i = (\Pi_i, \alpha_i)$, the IR prover forwards the plaintext message $\alpha_i$, and the parties commit to the proof oracle $\Pi_i$ by invoking the $\mathsf{VCR.Com}$ protocol. At the end of the interaction, when the IOR verifier wishes to query the prover's oracles, the IR prover and verifier invoke the $\mathsf{VCR.Rdx}$ opening reduction to reduce the opening claims to claims about $\mathcal{R}_{\mathsf{out}}$.

We show that, whenever the IOR satisfies standard round-by-round knowledge soundness notions [BMNW25; BCFW25], the resulting IR achieves a strong knowledge soundness guarantee called state-restoration knowledge soundness (SRKS). This is important because it allows us to invoke the second step of

our construction, namely applying the Fiat–Shamir transform [FS86] to the IR to obtain a NIRK. We do this by introducing a definition of round-by-round knowledge soundness for IRs, and showing that this implies SRKS. Along the way, we also develop sequential composition theorems for RBRKS IRs.

While our transformation resembles prior IOR-to-IR transforms [BMNW25], one difference is that our IR produces an additional output claim for the VCR relation $\mathcal{R}_{\mathsf{out}}$. In more detail, let the IOR reduce a relation $\mathcal{R}_1$ to $\mathcal{R}_2$. Then prior transforms produce an IR that reduces from (a committed version of) $\mathcal{R}_1$ to (a committed version of) $\mathcal{R}_2$. In contrast, our transform produces an IR that reduces the original claim to the cartesian product of $\mathcal{R}_{\mathsf{out}}$ and (a committed version of) $\mathcal{R}_2$. The difference is because the former approach fully resolves VC opening claims, while our approach reduces them to claims about $\mathcal{R}_{\mathsf{out}}$. This may seem like kicking the can down the road: the opening reduction may be cheaper than a full opening protocol, but now we have to prove an extra output claim. However, this perspective has many benefits if we carefully choose $\mathcal{R}_2$ and $\mathcal{R}_{\mathsf{out}}$. For instance, one can leverage efficient techniques for batching, accumulating, and proving these claims, which our constructions are going to do.

**Canopy: cheaper opening protocols via proximity claims.** We construct a hash-based VCR scheme that provides savings when opening at a (relatively) large batch of points.

**Theorem 1.2** (informal, Section 6)**.** *Let $h$ be a random oracle with output length $\sigma$. Let $\Sigma$ be a finite field with $|\Sigma| \geq 2^{\sigma}$, and let $\mathcal{C}$ be a linear code over $\Sigma$ with message length $k$ and block length $n$. For message length $N \geq k$, there exists a VCR scheme in the random oracle model with alphabet $\Sigma$ with the following efficiency:*

- *Commitment size: $\sigma + (\log k + 1) \cdot \lceil \log |\Sigma| \rceil$ bits.*
- *Proof size for opening $q$ claims: $O(q(\log N - \log k + 1) \cdot \sigma)$ bits.*
- *Commitment time: $O(N + N/2 + \cdots + k + 2n)$ hashes, plus the time to encode $\mathcal{C}$ and $O(k)$ field operations.*
- *Opening time: $O(q(\log N - \log k + 1))$ hash-value reads and $O(q)$ field operations.*
- *Verifying time: $O(q(\log N - \log k + 1))$ hashes and $O(q)$ field operations.*
- *Output relation: sumcheck-constrained proximity of a (committed) purported codeword to $\mathcal{C}$.*

The construction underlying the aforementioned theorem works as follows: when committing to a message $\boldsymbol{m}$ of length $N$, start constructing a Merkle tree as usual, but instead of continuing all the way to the root, stop at some height $\log N - \log k$. Then, encode the result $k$-sized vector of Merkle subtree roots (i.e., Merkle *caps*) using $\mathcal{C}$ to obtain a codeword $\boldsymbol{c}$, and commit to the latter via a Merkle tree. The root $\mathsf{rt}$ of the latter is the Canopy commitment.

The opening protocol for a set of $q$ points proceeds with the prover simply sending the Merkle paths for the leaves for those points up to the cap. The verifier checks these paths, but is left to verify that the claimed roots of these paths are indeed correct and consistent with $\boldsymbol{c}$. These claims are of the form "the $i$-th element of the message underlying $\boldsymbol{c}$ is $\mathsf{rt}_i$" and can be transformed into proximity claims about $\boldsymbol{c}$ via standard techniques.

**Smaller proofs and faster verification for hash-based SNARKs.** We use these ingredients to construct hash-based SNARKs with proof sizes and verification costs lower than prior schemes via two approaches:

- *Black-box IOP approach:* We use our transformation to turn any IOP for some NP relation $\mathcal{R}$ into an IR from $\mathcal{R}$ to the VCR output relation $\mathcal{R}_{\mathsf{out}}$. Then, we compose this IR with an interactive argument for the latter relation to obtain an end-to-end argument for NP. The latter can be made non-interactive via the Fiat–Shamir transform [FS86]. $\mathcal{R}_{\mathsf{out}}$ can be "simpler" than $\mathcal{R}$, and hence it is plausible that efficient standalone arguments for it can exist. For instance, when using Canopy as the VCR scheme, the output relation is a sumcheck-constrained proximity relation, for which we can get efficient arguments by compiling IOPs of Proximity (IOPPs) [BBHR18; ACFY24; ACFY25] via the BCS transform. This generic approach yields substantial savings for certain IOPs. In our experimental evaluation, combining a suitably

parameterized Canopy with an IOP based on the Bolt code [GNR26] yields arguments that are about 50% smaller than their counterparts obtained by using Merkle trees ($\sim$ 6 MB vs $\sim$ 12 MB), and which require the verifier to compute 3$\times$ fewer hashes. These savings are obtained with only a $\sim$5-15% overhead for the argument prover.

- *Exploiting IOP structure:* While the foregoing generic approach provides savings while being agnostic to the input IOP, exploiting knowledge of the IOP's construction can yield further savings. For Canopy in particular, we can exploit the structure of almost all modern IOPs. These are usually constructed as a composition of two components: an IOR from an NP claim to a proximity claim for some code $\mathcal{C}$, and an IOPP that proves the latter proximity claim. Our construction uses Canopy to construct an IR from the initial IOR, so the output relation of this IR is be the product of the (committed version of the) IOR's proximity relation and Canopy's output relation. If Canopy uses the *same* code $\mathcal{C}$, then this product is just $\mathcal{R}^2_{\mathsf{out}}$, so can then prove both proximity claims together with an argument obtained by applying the BCS transform to the aforementioned IOPP. This batching can yield great benefits. For an IOP based on the Reed–Solomon code, and using WHIR [ACFY25] as the IOPP, Canopy yields arguments $\sim$20% smaller than Merkle trees ($\sim$ 410 kB vs $\sim$ 500 kB), and the verifier performs 1.5$\times$ fewer hashes. These savings are obtained with only a $\sim$20% overhead for the argument prover.

The code for these experiments is available at `https://github.com/randomshuffle/canopy-system`.

**Faster hash-based PCD by accumulating VCR claims.** Incrementally-verifiable computation (IVC) [Val08], and its generalization in Proof-Carrying Data (PCD) [CT10] are powerful tools. As recalled in Section 1, a standard way of constructing IVC is via recursive composition of arguments or reductions, which requires the IVC prover to prove the satisfaction of the argument/reduction verifier. The cost of the latter is called "recursion overhead". The initial constructions used recursive composition of succinct arguments [BCCT13; BCTV14], and using recent SNARKs based on "optimal" IOPs [MZ25; BFRW26; BMMS26b] would yield hash-based IVC with state-of-the-art recursion overhead of $O(\lambda \log N)$ hashes. Unfortunately, as stated in the intro, this is concretely expensive, and it may be possible to do better asymptotically as well.

If we instead use recursive composition of certain non-interactive reductions via *accumulation* [BCMS20; BCL+21] or *folding* [KST22] schemes, then we can obtain better concrete efficiency, but unfortunately still does not lead to asymptotic improvements. We improve this state-of-affairs by using our VCR schemes to design a new accumulation scheme with improved asymptotic efficiency:

**Theorem 1.3** (informal, Section 8.3)**.** *There exists a hash-based accumulation scheme for NP with prover time $O(N)$ verifier time $O(\lambda\sqrt{\log N})$ hashes, and decider time $O(N)$.*

We obtain this result by adapting existing constructions of hash-based accumulation schemes from hash-based IRs (which use Merkle trees [BMNW25]) to instead work with VCR-based IRs produced by our transformation. The high level idea behind the savings is that while in the prior approach the accumulation verifier fully checked VC opening claims, in our approach, the verifier instead produces new claims about the VCR scheme's output relation $\mathcal{R}_{\mathsf{out}}$. If one has an accumulation scheme for these claims, then one does not have to check these claims on the spot; instead, they can just be accumulated as well. Using this idea with Canopy yields the foregoing theorem since there exist efficient accumulation schemes [BMNW25; BCFW25; BMMS26b] for Canopy's output relation.

We also obtain concrete improvements over state-of-the-art accumulation schemes based on Reed–Solomon code [BMNW25]. For accumulating 8 new statements, a Canopy based accumulation scheme requires less than 50% verifier hashes at the cost of 40% extra prover time compared to a rate 1/2 Reed–Solomon baseline.

## 1.2 Related work

### 1.2.1 Methodologies for constructing arguments

**IORs to hash-based reductions via Merkle trees.** The very first works on succinct arguments [Kil92; Mic00] already used Merkle trees to compile non-interactive Probabilistically Checkable Proofs (PCPs) to succinct arguments. Subsequently, this approach was generalized by [BCS16] to work with interactive protocols, namely Interactive Oracle Proofs (IOPs). Subsequently, [BMNW25] further generalized the transformation of [BCS16], by presenting a transformation from IORs to NIRKs (Non-Interactive Reductions of Knowledge) via the use of Merkle trees and Fiat–Shamir, also proving their transformation secure in the ROM.

**PCPs to hash-based arguments via Merkle tree variants.** In addition to the foregoing Merkle-tree-based transformations, there has also been recent work [CY21] on transforming PCPs into hash-based arguments *without* relying on Merkle trees. At a high level, this work departs from the blueprint of using vector commitments to commit to the oracles in a PCP. It instead shows that PCPs satisfying a certain property (which happens to already be satisfied by existing PCPs of interest) can be transformed into hash-based arguments using commitment schemes that are instead *weakly binding*, i.e. a commitment does not bind a prover to exactly one vector. The resulting hash-based SNARKs have smaller proof sizes than can would be obtained by using fully binding hash-based vector commitments, i.e. Merkle trees, to commit to the oracles in the PCP. Extending this approach to IOPs and IORs remains an interesting open direction. Another interesting direction is to explore whether these techniques can be combined with our VCR-based techniques to obtain hash-based SNARKs with even smaller proofs.

**IORs to arguments based on algebraic VC schemes.** In the groups of unknown order setting (which is not post-quantum secure), [LM19] and [BBF19] study how to transform PCPs and IOPs, respectively, into SNARKs using vector commitments whose opening proofs consist of a constant number of group elements.

**FIOPs to SNARKs via functional commitments.** A large body of work considers IOPs whose oracles carry additional algebraic structure, most notably polynomial IOPs (PIOPs), and compiles them into SNARKs using commitment schemes tailored to that structure (which are often not solely hash-based), most notably polynomial commitment schemes (see, e.g., [MBKM19; GWC19; CHM+20; BFS20; CBBZ23]). This paradigm is unified in [CGKY26], which captures the foregoing notions by defining functional IOPs (FIOPs) and functional commitment (FC) schemes, respectively. Furthermore, [CGKY26] present Funky, a transformation that combines an FIOP with a compatible FC scheme, and subsequently applies the Fiat–Shamir transformation, to obtain a SNARK. The security analysis of Funky differs from ours, relying on a notion of state-restoration function binding—the functional analogue of position binding—and a rewinding-based extraction strategy, allowing them to accommodate non-hash-based FC schemes. Extending such a security analysis to VCR schemes and our VCR-based transformations is an interesting direction for future work.

### 1.2.2 Alternative Merkle tree constructions

**Verkle trees** [Kus18] generalize Merkle trees by replacing the hashes at each node with vector commitments. In this construction, the authentication paths no longer contain sibling hashes; instead each node contains a VC opening proof that proves the value of its $i$-th child. Using high-arity trees, this approach can lead to great proof-size savings if the VC opening proofs are small enough. Unfortunately, in the ROM, the only known option for a VC scheme is a Merkle tree, which results in the entire construction being a Merkle tree itself.
**Coded merkle trees** [YSL+20] employ a similar idea to ours: encoding intermediate layers of a Merkle tree with error-correcting codes. They do not construct any vector-commitment-like primitive from their construction, and instead use it for providing data availability guarantees in decentralized systems.

# 2 Techniques

In Section 2.1, we begin with a simplified definition of VCR schemes and the properties that we require of them (see full definition in Section 4). In Section 2.2, we then describe a simplified IOR-to-IR compiler using arbitrary VCR schemes (see full construction in Section 5). In Section 2.3, we informally describe Canopy, our VCR scheme that uses high-distance codes to decrease opening proof sizes (see Section 6 for formal definition). Finally, in Section 2.4, we describe how Canopy can be used to construct SNARKs and accumulation schemes with smaller proof sizes (see Sections 7 and 8 for details).

## 2.1 Knowledge soundness of VCR schemes

We elaborate on the informal definition of VCR schemes provided in Definition 1.1, with the end goal of providing intuition for strong knowledge soundness guarantees that we require. Let $\mathsf{VCR} = (\mathsf{Com}, \mathsf{Rdx}, \mathcal{R}_{\mathsf{out}})$ be a VCR scheme. Here, we have

- $\mathsf{Com}$ is an interactive protocol between the prover and verifier. The prover, on input a message $\boldsymbol{m}$, interacts with the verifier over a number of rounds. At the end of the interaction, the verifier outputs the protocol transcript as the commitment $\mathsf{cm}$, while the prover outputs an opening trapdoor $\mathsf{td}$. The latter is information that helps the prover during the opening reduction. We will denote the set of "valid" commitment-preimage pairs via the relation $\mathcal{R}_{\mathsf{com}} \subset \{(\mathsf{cm}, (\boldsymbol{m}, \mathsf{td}))\}$.
- $\mathsf{Rdx}$ is an interactive reduction. The prover and verifier interact to reduce opening claims at a query set $Q$ to claims about $\mathcal{R}_{\mathsf{out}}$. The opening claims are for the following query relation:

$$\mathsf{Query}[\mathcal{R}_{\mathsf{com}}] := \{((\mathsf{cm}, Q, \boldsymbol{a}), (\boldsymbol{m}, \mathsf{td})) \mid (\mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \mathcal{R}_{\mathsf{com}} \wedge \boldsymbol{m}[Q] = \boldsymbol{a}\}$$

Before we define knowledge soundness notions for VCR, we introduce one more relation, $\mathcal{R}_{\mathsf{bad}} \subset \{(\mathsf{cm}, \mathsf{td})\}$, that consists of commitments and trapdoors which are not binding (e.g., because the commitment protocol verifier sampled a bad challenge). VCR must satisfy the following properties:

1. *Property 1: Valid commitments are binding.* If a commitment has a valid opening (i.e., there exists a message-trapdoor pair $(\boldsymbol{m}, \mathsf{td})$ such that $(\mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \mathcal{R}_{\mathsf{com}}$), then the probability (over the choice of the verifier's coins) that $\mathsf{cm}$ is not binding (i.e., $(\mathsf{cm}, \mathsf{td}) \in \mathcal{R}_{\mathsf{bad}}$) is negligible.

2. *Property 2: Valid commitments are extractable.* There is an efficient deterministic extractor $\mathcal{E}_{\mathsf{Com}}$ with the following guarantee. Consider an efficient two-stage adversary that first outputs a commitment $\mathsf{cm}$, and subsequently outputs an opening $(\boldsymbol{m}, \mathsf{td})$. Let $\mathsf{qtr}_0$ be its random-oracle query trace through the first stage, and write $(\boldsymbol{m}', \mathsf{td}') \leftarrow \mathcal{E}_{\mathsf{Com}}(\mathsf{cm}, \mathsf{qtr}_0)$. The probability that the adversary's opening is valid, but either $(\mathsf{cm}, (\boldsymbol{m}', \mathsf{td}')) \notin \mathcal{R}_{\mathsf{com}}$ (i.e., the commitment-opening pair is invalid), or $(\mathsf{cm}, \mathsf{td}') \notin \mathcal{R}_{\mathsf{bad}}$ and $\boldsymbol{m}' \neq \boldsymbol{m}$, (i.e., the commitment is binding yet the extracted message is inconsistent with the adversarial one), is negligible.

3. *Property 3: Reduction extractability.* The opening reduction $\mathsf{VCR.Rdx}$ is knowledge sound. There is an efficient extractor $\mathcal{E}_{\mathsf{Rdx}}$ such that the following event has negligible probability: an adversarial prover makes the opening verifier accept a claim $(\mathsf{cm}, Q, \boldsymbol{a})$ and supplies a valid witness for the resulting $\mathcal{R}_{\mathsf{out}}$ instance, but the extractor fails to recover $(\boldsymbol{m}, \mathsf{td})$ satisfying $(\mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \mathcal{R}_{\mathsf{com}}$ and $\boldsymbol{m}[Q] = \boldsymbol{a}$. The extractor receives the execution transcript, the relevant random-oracle query traces, and the supplied output witness.

For simplicity in the technical overview, we omit the following details that the full definition in Section 4 needs to account for:

- Partial strings: the extractor does not necessarily have to output an exact pre-image $\boldsymbol{m}$ for a commitment $\mathsf{cm}$. $\boldsymbol{m}$ may be a partial string, in that some of its coordinates may be undefined (denoted $\perp$). In that event the extractor only needs to output a $\boldsymbol{m}'$ that is a super-string of $\boldsymbol{m}$ (so $\boldsymbol{m}'$ can also be a partial string).

- Promise relations: $\mathcal{R}_{\mathsf{com}}$ is in fact a tuple of two relations $(\mathcal{R}_{\mathsf{com}}, \tilde{\mathcal{R}}_{\mathsf{com}})$ such that $\mathcal{R}_{\mathsf{com}} \subseteq \tilde{\mathcal{R}}_{\mathsf{com}}$. The former is a strict relation that is used to define completeness, while the latter is a relaxed relation that is used to define soundness, binding and extraction.

- Round-by-round sparsity of $\mathcal{R}_{\mathsf{bad}}$: we not only require valid commitments being in $\mathcal{R}_{\mathsf{bad}}$ to be unlikely, but we require the event of a partial commitment going from being not in $\mathcal{R}_{\mathsf{bad}}$ to being in $\mathcal{R}_{\mathsf{bad}}$, to be unlikely *in each round of the commitment protocol* (empty commitments are not in $\mathcal{R}_{\mathsf{bad}}$). This property allows us to prove round-by-round knowledge soundness of the compiled interactive reduction when oracles are instantiated with a VCR scheme.

## 2.2 From IORs to IRs via VCR schemes

We now elaborate on how we construct interactive reductions (IRs) from interactive oracle reductions (IORs) via VCR schemes. Let $(\mathbf{P}, \mathbf{V})$ be an IOR with $k$ rounds from $\mathcal{R}$ to $\mathcal{R}'$. In each round, the prover sends $M_i = (\Pi_i, \alpha_i)$, where $\Pi_i$ is an oracle string of length $\ell$ over an alphabet $\Sigma$ and $\alpha_i$ is a plaintext message. The verifier receives oracle access to $\Pi_i$ and explicit access to $\alpha_i$, and responds with a uniformly random challenge $\rho_i$ in $\Sigma$. At the end, the verifier queries each oracle, and emits an instance for a different relation $\mathcal{R}'$, while the prover emits the corresponding witness.

Let $\mathsf{VCR} = (\mathsf{Com}, \mathsf{Rdx}, \mathcal{R}_{\mathsf{out}})$ be a VCR scheme. An IR from $\mathcal{R}$ to $\mathcal{R}' \times \mathcal{R}_{\mathsf{out}}^k$ can be constructed from this IOR and VCR scheme as follows. For each round $i \in [k]$, the IR prover invokes $\mathbf{P}$ to obtain $M_i = (\Pi_i, \alpha_i)$ and forwards $\alpha_i$ to the IR verifier. The IR prover and verifier then invoke $\mathsf{VCR.Com}$ to obtain a commitment $\mathsf{cm}_i$ to this oracle. After the $k$ IOR rounds have concluded, the IR prover invokes the IOR verifier to determine its queries $Q_1, \ldots, Q_k$ to each oracle. It then computes the corresponding responses $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k$, and sends these to the IR verifier. The two then invoke $\mathsf{VCR.Rdx}$ to reduce these opening claims to $k$ instances of $\mathcal{R}_{\mathsf{out}}$. This completes the description of the IR.

Completeness follows straightforwardly from that of the ingredients, but proving knowledge soundness is more involved, so we describe the proof below.

**Lemma 2.1** (informal)**.** *Let $(\mathbf{P}, \mathbf{V})$ be a knowledge-sound IOR, and let $\mathsf{VCR}$ be a VCR scheme satisfying the knowledge soundness guarantees from Section 2.1. Then the IR constructed above is knowledge sound.*

**Proof sketch.** Let $\widetilde{\mathcal{P}}_{\mathsf{IR}}$ be an adversarial prover against the IR. We use it to construct an adversarial prover $\widetilde{\mathcal{P}}_{\mathsf{IOR}}$ against the IOR that works as follows:

1. For each IOR round $i$:
   (a) Receive the plaintext message $\alpha_i$ from $\widetilde{\mathcal{P}}_{\mathsf{IR}}$ and simulate the commitment protocol for oracle $\Pi_i$, obtaining commitment $\mathsf{cm}_i$.
   (b) Invoke $\mathcal{E}_{\mathsf{Com}}$ with input $\mathsf{cm}_i$ to extract an underlying oracle $\Pi_i'$ and trapdoor $\mathsf{td}_i'$, and send the prover message $M_i' = (\Pi_i', \alpha_i)$ to $\mathbf{V}$.
   (c) Receive $\rho_i$ in response, and forward it to $\widetilde{\mathcal{P}}_{\mathsf{IR}}$.
2. Use $\alpha_i$, challenges $\rho_i$, and extracted oracles to compute $\mathbf{V}$'s queries, and send them to $\widetilde{\mathcal{P}}_{\mathsf{IR}}$.
3. Invoke $\mathcal{E}_{\mathsf{Rdx}}$ with opening adversary $\widetilde{\mathcal{P}}_{\mathsf{Rdx}}$ constructed above. The extractor will output an oracle $\Pi_i''$ and trapdoor $\mathsf{td}_i''$ for each $i \in [k]$.
4. Assert that, for each $i \in [k]$, $\Pi_i'$ is consistent with $\Pi_i''$.

5. At the end of the interaction, $\widetilde{\mathcal{P}}_{\mathsf{IR}}$ outputs witnesses for $\mathcal{R}'$ and $\mathcal{R}_{\mathsf{out}}$. Assert that the latter witness is valid.
6. Output the witness for $\mathcal{R}'$.

Let $V$ denote the event that the IR verifier accepts, and that the output witness is valid for $\mathcal{R} \times \mathcal{R}_{\mathsf{out}}^k$. Let $O_i$ denote the event that the $i$-th invocation of VCR.Rdx verifier accepts, and that this invocation produces a valid witness in $\mathcal{R}_{\mathsf{out}}$. Clearly, $V$ happens only if $O_i$ happens for each $i$.

We argue that $\widetilde{\mathcal{P}}_{\mathsf{IOR}}$ is successful whenever $\widetilde{\mathcal{P}}_{\mathsf{IR}}$ is successful, up to failure of the VCR extractors.

First, we will ensure that Step 3 succeeds. We define adversaries $\widetilde{\mathcal{P}}_{\mathsf{Rdx},i}$ that $\mathcal{E}_{\mathsf{Rdx}}$ must extract from. $\widetilde{\mathcal{P}}_{\mathsf{Rdx},i}$ receives as input the interaction transcript up to the $i$-th opening reduction, and the corresponding random-oracle query trace. It simulates $\widetilde{\mathcal{P}}_{\mathsf{IR}}$ for all $k$ IOR rounds and the first $i-1$ opening reductions using the transcript. To simulate $\widetilde{\mathcal{P}}_{\mathsf{IR}}$'s interaction for the $i$-th execution of VCR.Rdx, $\widetilde{\mathcal{P}}_{\mathsf{Rdx},i}$ uses the challenges it receives from its corresponding VCR.Rdx verifier. When $\widetilde{\mathcal{P}}_{\mathsf{IR}}$ outputs its witness for $\mathcal{R}_{\mathsf{out}}$, $\widetilde{\mathcal{P}}_{\mathsf{Rdx},i}$ outputs this and halts. Clearly the view of the simulated $\widetilde{\mathcal{P}}_{\mathsf{IR}}$ is identical to that of the real $\widetilde{\mathcal{P}}_{\mathsf{IR}}$, and hence $\widetilde{\mathcal{P}}_{\mathsf{Rdx},i}$'s messages are distributed identically to those of (the relevant parts of) $\widetilde{\mathcal{P}}_{\mathsf{IR}}$.

Denote by $T_i$ the event that the extracted $(\Pi_i'', \mathsf{td}_i'')$ is a valid witness for $(\mathsf{cm}_i, Q_i)$. Since VCR.Rdx is knowledge-sound by Property 3, we know that the probability that $O_i$ happens, but $T_i$ does *not* happen is negligible. Hence, the probability that $V$ happens but $T_i$ does not happen for some $i$ is also negligible. That is, as long as the IR verifier succeeds, it must be the case (except with negligible probability), that the extracted $(\Pi_i'', \mathsf{td}_i'')$ pair is valid. Recall that the witness being valid here means that $(\mathsf{cm}_i, (\Pi_i'', \mathsf{td}_i'')) \in \mathsf{VCR}.\mathcal{R}_{\mathsf{com}}$.

Next, we argue that Step 1b succeeds. We define two-stage adversaries $\widetilde{\mathcal{P}}_{\mathsf{Com},i}$ that $\mathcal{E}_{\mathsf{Com}}$ must extract from. In the first stage, the adversary simulates $\widetilde{\mathcal{P}}_{\mathsf{IR}}$'s execution up to, and including, the computation of commitment $\mathsf{cm}_i$ via the input transcript. This interaction produces $\mathsf{cm}_i$, which it outputs. It then stores $\widetilde{\mathcal{P}}_{\mathsf{IR}}$'s state for the second stage. In the latter, it continues the same execution, and eventually outputs $(\Pi_i'', \mathsf{td}_i'')$ by invoking the $\mathcal{E}_{\mathsf{Rdx}}$ as above (these are the *same* as the output of the VCR.Rdx extractor since they are both obtained from the IR adversary's output). Once again, the views of $\widetilde{\mathcal{P}}_{\mathsf{IR}}$ in this simulation are identical to those in the real execution, and hence $\widetilde{\mathcal{P}}_{\mathsf{Com},i}$'s messages are distributed identically to those of $\widetilde{\mathcal{P}}_{\mathsf{IR}}$.

Commitment extractability (see Property 2) gives us the guarantee that whenever the second stage of $\widetilde{\mathcal{P}}_{\mathsf{Com},i}$ outputs a valid opening of $(\Pi_i'', \mathsf{td}_i'')$, then, with all but negligible probability, the extractor's output $(\Pi_i', \mathsf{td}_i')$ must also be a valid opening, and moreover, this opening must either be bad ($(\mathsf{cm}_i, \mathsf{td}_i) \in \mathcal{R}_{\mathsf{bad}}$), or agree with the adversary's message ($\Pi_i' = \Pi_i''$). Since the above argument ensures that $(\Pi_i'', \mathsf{td}_i'')$ is valid, and since valid commitments are bad (non-binding) only with negligible probability by Property 1, it must be the case that extraction failure can only happen when $\Pi_i' \neq \Pi_i''$. Since commitment extractability guarantees that extraction failure happens with negligible probability, we know that with overwhelming probability, $\Pi_i' = \Pi_i''$.

The constructed IOR prover's success probability is therefore at least that of $\widetilde{\mathcal{P}}_{\mathsf{IR}}$, minus the three different VCR-related error bounds.

Finally, consider the extractor $\mathcal{E}_{\mathsf{IR}}$ that simply invokes the IOR extractor against $\widetilde{\mathcal{P}}_{\mathsf{IOR}}$ constructed from $\widetilde{\mathcal{P}}_{\mathsf{IR}}$, and outputs its witness for $\mathcal{R}$. Since the IOR is knowledge-sound, we know that this step cannot fail, and hence $\mathcal{E}_{\mathsf{IR}}$ successfully extracts from $\widetilde{\mathcal{P}}_{\mathsf{IR}}$ if it succeeds.

### 2.3 Canopy: a VCR scheme based on error-correcting codes

IOPs often make a large number of queries to each oracle to achieve a small soundness error. These queries become expensive when we transform the IOP into a hash-based argument. When an oracle of length $n$ is committed with a Merkle tree, each query requires an authentication path containing $\log n$ hash values. For $B$ queries, these paths contribute $B \log n$ hash values to the proof, before accounting for shared nodes. Canopy reduces this cost by shortening the paths while keeping the commitment small.

**Short paths and a large cap.** A natural first attempt is to stop building the Merkle tree at height $d$ above its leaves. This partitions the oracle into blocks of $2^d$ entries, each with its own Merkle root. We refer to the resulting vector of $k = n/2^d$ roots as a *cap*. If the verifier knows the entire cap, each queried entry can be checked against its block root using a path of only $d$ hash values. The paths now cost $Bd$ hash values, but the commitment contains all $k$ cap roots. For small $d$, sending this cap is far more expensive than sending a single root. Therefore, we need a shorter commitment to the cap that still lets us check all the queried $B$ leaves.

Let's assume that we have a polynomial commitment scheme. Choose a field large enough to represent each hash value as a distinct field element, using a fixed injective map $\mathsf{H2F}$. Applying this map to the cap gives a vector $\boldsymbol{y} \in \mathbb{F}^k$. Index its entries by $v \in \{0,1\}^{\log k}$ and let $\hat{y}$ be its multilinear extension, so that

$$\hat{y}(v) = \boldsymbol{y}[v] = \mathsf{H2F}(\mathsf{rt}_v) \qquad \text{for every } v \in \{0,1\}^{\log k}.$$

We replace the cap with a commitment to $\hat{y}$. To open an entry of the original oracle, the prover sends its block root and the path to that root, then proves the corresponding evaluation of $\hat{y}$. For $B$ queries that touch $t \leq B$ blocks, the proof contains $Bd$ path hashes, $t$ block roots, and proofs of $t$ polynomial evaluations. However, we can combine these $t$ evaluation claims into one by interpreting them as a single *sumcheck* claim.

**One claim for all the cap entries.** Let $v_1, \ldots, v_t$ be the touched block indices. After receiving the claimed roots, the verifier samples a random field element $\gamma$ to combine their evaluation claims. Taking a random linear combination of these sums gives the single claim

$$\sum_{\boldsymbol{b} \in \{0,1\}^{\log k}} \hat{y}(\boldsymbol{b}) \left( \sum_{i=1}^{t} \gamma^i \cdot \mathsf{eq}(\boldsymbol{b}, v_i) \right) = \sum_{i=1}^{t} \gamma^i \cdot \mathsf{H2F}(\mathsf{rt}_{v_i}) \ , \tag{1}$$

where eq is the multilinear equality polynomial, which on Boolean inputs is one when the inputs agree and zero otherwise. By Schwartz-Zippel, except with small probability over $\gamma$, all the evaluation claims must be true if the combined claim is true. We then use the sumcheck protocol to reduce (1) to one evaluation of $\hat{y}$ at a random point, using $O(\log k)$ field elements of communication. Finally, we use the polynomial commitment scheme to check that final evaluation. Thus, the overall proof now consists of the query answers, the $Bd$ hash values, a sumcheck proof, and one polynomial evaluation proof.

**Building the polynomial commitment.** We realize this last ingredient in the natural way for hash-based schemes: using an error-correcting code and another Merkle tree. First, we encode the cap vector $\boldsymbol{y}$ with a linear code $\mathcal{C}$ of message length $k$ and block length $b$. The prover commits to the resulting word $\boldsymbol{c} = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y})$ using a Merkle tree with root $\mathsf{rt}$. This root replaces the large cap, resulting in the illustration in Fig. 1.

Now, the verifier needs to check that the word committed under $\mathsf{rt}$ is close to the encoding of a cap vector whose multilinear extension satisfies Eq. (1). This is a *sumcheck-evaluation proximity* claim, or $\mathsf{SEP}$ claim, for the code $\mathcal{C}$. It combines the sumcheck equation with proximity to an encoding, as defined in Section 3.2. An honest prover commits to the exact encoding, while soundness allows a word within a specified relative distance $\delta$ of it. We can prove this claim with a code-based proximity protocol, with its oracle queries authenticated under the top Merkle root. Thus, the full opening consists of the short paths and touched cap roots, plus a proof of this $\mathsf{SEP}$ claim.

**Why the new proof can be smaller.** Now, it may seem like we have not made progress, and simply replaced the savings of $B \cdot (\log n - d)$ hashes with the cost of proving a $\mathsf{SEP}$ claim. The new proof still involves Merkle paths in the top tree, so the gain depends on making much fewer than $B$ queries there. The key is that the message encoded by $\mathcal{C}$ has length $k = n/2^d$, rather than $n$. We can therefore choose a code with a much smaller rate $\rho = k/b$ while keeping its encoded length $b$ no larger than $n$ (so as to not incur a large

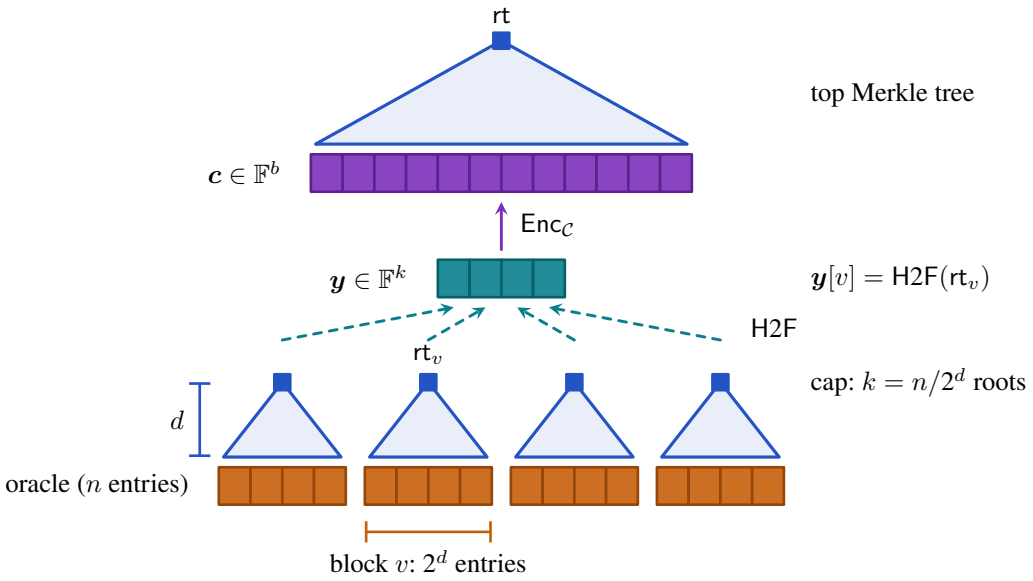


**Figure 1:** The two layers of Canopy, illustrated with $n = 16$ oracle entries, depth $d = 2$, and $k = 4$ blocks. Each block (orange) of $2^d$ entries has a depth-$d$ Merkle tree (blue) with root $\mathsf{rt}_v$. The cap of $k = n/2^d$ roots is mapped into $\mathbb{F}$ by $\mathsf{H2F}$ to form $\boldsymbol{y} \in \mathbb{F}^k$ (teal). Its encoding $\boldsymbol{c} = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y}) \in \mathbb{F}^b$ (purple) is committed with the top Merkle tree rooted at $\mathsf{rt}$.

penalty in prover time). A smaller rate translates to a larger relative distance, which allows for a larger $\delta$ in the aforementioned proximity protocol, which then translates to fewer queries to achieve a given soundness error. We choose $d$ to balance those paths against the $Bd$ hashes in the lower trees, while also accounting for the cost of encoding the cap. This gives concrete proof-size savings for SNARKs and asymptotic hash-cost savings for accumulation schemes, as detailed in Sections 7.2 and 8.3.

**Binding to one cap vector.** The VCR scheme definition is in part motivated by our Canopy construction. In particular, two features of our VCR definition are necessary to make Canopy work. First, to take full advantage of relative distances approaching 1, we must allow $\delta$ to lie in the list-decoding regime, where there exist multiple codewords within distance $\delta$ of a given word. The top Merkle root binds the prover to a word, but proximity to that word alone may not select a single cap vector, which in turn may lead to multiple conflicting openings of the same input message.

Following STIR [ACFY24], we resolve this by using an *out-of-domain evaluation*. After the prover sends $\mathsf{rt}$, the verifier samples $\boldsymbol{z}_{\mathsf{ood}} \in \mathbb{F}^{\log k}$, and the prover replies with $\zeta_{\mathsf{ood}} = \hat{y}(\boldsymbol{z}_{\mathsf{ood}})$. For a sufficiently large field and a bounded decoding list, the multilinear extensions of all candidate cap vectors have distinct evaluations at the out-of-domain point except with small probability. Thus, the prover's reply binds it to a single cap vector, which in turn binds it to a single input message. Binding to a single message is established only after this interaction, which is why we allow the VCR commitment protocol to be interactive. For Canopy, the complete commitment is the transcript $(\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}})$. Note that the out-of-domain reply is an evaluation claim that must also be checked, but can be easily batched into the $\mathsf{SEP}$ claim for the cap evaluations.

**Leaving the claim for later.** The second feature is that the opening reduction can stop once it has produced the $\mathsf{SEP}$ claim: it does not need prove the $\mathsf{SEP}$ claim immediately, and the verifier only checks the short paths immediately. Several IOPs already reduce their own checks to $\mathsf{SEP}$ claims, and accumulation schemes maintain related claims across successive steps. By choosing the Canopy code to match these claims, we

can combine the claims arising from openings with those that the surrounding protocol already needs to handle. For SNARKs, we prove the resulting batch together using the protocol of Appendix D, sharing the sumcheck and proximity proof across the claims. For accumulation schemes, we combine the new claims with the existing accumulator and defer their final check, as in Section 8. Canopy thus reduces vector openings to claims that the surrounding protocol can batch, prove, or accumulate.

### 2.4 Constructing SNARKs and accumulation schemes from VCR schemes

We discuss how we use the VCR-based IOR-to-IR transformation from Section 2.2 with Canopy as the underlying VCR scheme to obtain improved hash-based SNARKs and accumulation schemes.

**Hash-based SNARKs.** As discussed in Section 1.1, our approach can be used to transform any given IOP for a relation $\mathcal{R}$ using any given VCR scheme into a hash-based SNARK for $\mathcal{R}$ as follows: (i) instantiate the oracles using the VCR scheme, (ii) use an interactive argument to prove all the claims output by the VCR opening reductions, and (iii) use the Fiat-Shamir transformation to make the protocol non-interactive. We discuss the details of a generalization of this generic approach in Section 7.1.

In Section 7.2, we present a more optimized Canopy-based approach to compile a special class of IOPs, those constructed using IOPPs (which includes all state-of-the-art IOPs), into hash-based SNARKs. The optimized approach exploits the fact that IOPP-based IOPs already involve proving $\mathsf{SEP}[\mathcal{C}]$ claims. Therefore, by parametrization Canopy such that its output claims are also $\mathsf{SEP}[\mathcal{C}]$ claims, one can leverage existing batching techniques to batch-prove both $\mathsf{SEP}[\mathcal{C}]$ claims.

**Hash-based accumulation schemes.** At a high level, an accumulation scheme $\mathsf{ACC}$ for $\mathcal{R}$ allows one to reduce the task of checking $\ell_{\mathsf{acc}}$ membership claims for $\mathcal{R}$ to checking a single accumulated claim.[2]

Prior work [BMNW25; BCFW25; BMMS26a] has constructed hash-based accumulation schemes using the following simplified blueprint: first, construct a corresponding IOR that accumulates claims with respect to $\ell_{\mathsf{acc}}$ input oracles of length $n$ (which roughly corresponds to the size of a tuple in $\mathcal{R}$), where the IOR verifier makes $O(\lambda)$ queries to each of the input oracles. Then, transform this IOR into a non-interactive hash-based accumulation schemes by replacing the oracles with Merkle commitments and turning the protocol non-interactive using the Fiat-Shamir transformation. This results in the resulting accumulation verifier performing $O(\lambda \ell_{\mathsf{acc}} \log n)$ hashes: $O(\lambda)$ queries are made to each of the $\ell_{\mathsf{acc}}$ oracles, and each query results in a Merkle authentication path of $O(\log n)$ hashes.

However, we can commit to all the oracles of the IOR with Canopy instead of Merkle trees obtain a hash-based accumulation scheme with better efficiency. In particular, we can parameterize Canopy as follows:

- Set Canopy depth $d = \sqrt{\log n}$. In this case, the verifier needs to make only $O(\lambda\sqrt{\log n})$ hashes to verify the opening reduction corresponding to the $O(\lambda)$ queries made to each of the $\ell_{\mathsf{acc}}$ Canopy-committed oracles.
- Set the Canopy code $\mathcal{C}$ to have distance $1 - 2^{-\sqrt{\log n}}$. For example, this can be achieved if $\mathcal{C}$ is a Reed–Solomon code with rate $2^{-\sqrt{\log n}}$ and thus block length $n$, the same as the length of the input oracle. The opening reduction outputs $\ell_{\mathsf{acc}}$-many $\mathsf{SEP}[\mathcal{C}]$ claims, which can be batched into one $\mathsf{SEP}[\mathcal{C}]$ claim by making $O(\lambda/\sqrt{\log n})$ queries to each oracle. Instantiating these oracles with Merkle commitments, the verifier only needs to compute $O(\log n)$ hashes for each query, and thus, $O(\lambda\sqrt{\log n})$ hashes for each of the $\ell_{\mathsf{acc}}$ oracles.

We present the full details of our Canopy-based accumulation scheme in Section 8.

[2] The actual definition enables one to batch $\ell_{\mathsf{acc},1}$ membership checks for $\mathcal{R}$ along with $\ell_{\mathsf{acc},2}$ accumulated claims (produced at some point in the past), into a single accumulated claim. We use the simplified definition here for ease of exposition.

# 3 Preliminaries

For a positive integer $n$, we write $[n] \coloneqq \{1, \ldots, n\}$. For a vector $\boldsymbol{x} \in \Sigma^\ell$ and set of indices $S \subseteq [\ell]$, $\boldsymbol{x}[S]$ denotes the vector obtained by restricting $\boldsymbol{x}$ to the coordinates in $S$. We also consider vectors over partial alphabets $\Sigma \cup \{\bot\}$, and use $\mathrm{Dom}(\boldsymbol{x})$ to denote the set of indices $i$ such that $\boldsymbol{x}[i] \neq \bot$. For partial strings $\boldsymbol{m}, \boldsymbol{m}' \in (\Sigma \cup \{\bot\})^\ell$, write $\boldsymbol{m} \preceq \boldsymbol{m}'$ if, for every $j \in [\ell]$, either $\boldsymbol{m}[j] = \bot$ or $\boldsymbol{m}[j] = \boldsymbol{m}'[j]$. Thus, $\boldsymbol{m}'$ extends $\boldsymbol{m}$ on every coordinate where $\boldsymbol{m}$ is defined. We use $x \xleftarrow{\$} S$ to denote that $x$ is sampled uniformly at random from a finite set $S$.

**Hamming distance.** For vectors $\boldsymbol{x}, \boldsymbol{y} \in (\Sigma \cup \{\bot\})^n$, their fractional Hamming distance is

$$\delta(\boldsymbol{x}, \boldsymbol{y}) \coloneqq |\{i \in [n] : \boldsymbol{x}[i] \neq \boldsymbol{y}[i] \vee \boldsymbol{x}[i] = \bot \vee \boldsymbol{y}[i] = \bot\}|/n \ .$$

We write $\boldsymbol{x} \sim_\delta \boldsymbol{y}$ when $\delta(\boldsymbol{x}, \boldsymbol{y}) < \delta$.

**Random oracles.** For $r \in \mathbb{N}$, let $\mathcal{U}(r)$ denote the uniform distribution over functions $\{0,1\}^* \to \{0,1\}^r$; thus $h \xleftarrow{\$} \mathcal{U}(r)$ is a random oracle with $r$-bit outputs. Bit-string-valued random oracles of varying output lengths can be derived from a single one via domain separation and output extension [CY24, Section 7.4]. More generally, for a finite nonempty set $S$, let $\mathcal{U}(S)$ denote the uniform distribution over functions $\{0,1\}^* \to S$.

**Definition 3.1** (query-bounded ROM algorithm)**.** A *$q$-query ROM algorithm* is a randomized algorithm with query access to a random oracle $h \xleftarrow{\$} \mathcal{U}(\sigma)$ that makes at most $q$ random-oracle queries. It may adaptively query $h$ and retain state. Unless stated otherwise, a ROM algorithm may be computationally unbounded. The *random-oracle trace* of a ROM algorithm $\mathcal{A}$, denoted $\mathtt{qtr}$, is an ordered tuple of all queries and responses to the random oracle.

## 3.1 Relations

**Indexed relations.** An *indexed relation* $\mathcal{R}$ is a set of triples $\{(\mathtt{i}, \mathtt{x}, \mathtt{w})\}$ where $\mathtt{i}$ is the index, $\mathtt{x}$ is the instance, and $\mathtt{w}$ is the witness; the corresponding *indexed language* $\mathcal{L}(\mathcal{R})$ is the set of pairs $(\mathtt{i}, \mathtt{x})$ for which there exists a witness $\mathtt{w}$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$. We use $\mathcal{R}(\mathtt{i})$ to denote the set of all tuples $(\mathtt{x}, \mathtt{w})$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$.

**Promise relations.** Some proof systems exhibit a gap between completeness and soundness. This is formalized by using promise relations of the form $\mathcal{R} = (\mathcal{R}, \tilde{\mathcal{R}})$, where completeness holds for a *strict* relation $\mathcal{R}$, but soundness only guarantees membership in a *relaxed* relation $\tilde{\mathcal{R}} \supseteq \mathcal{R}$.

**Indexed oracle promise relations.** An *indexed oracle promise relation* is an indexed promise relation whose elements consist of a tuple of instance oracles strings $\vec{\mathtt{y}} = (\mathtt{y}_1, \ldots, \vec{\mathtt{y}}_{|\vec{\mathtt{y}}|})$ in addition to the (short and explicit) instance $\mathtt{x}$. We write tuples in an indexed oracle relation as

$$(\mathtt{i}, \mathtt{x}, \vec{\mathtt{y}}, \mathtt{w}) \in \mathcal{R} \ \ , \ \ \vec{\mathtt{y}} = (\mathtt{y}_1, \ldots, \mathtt{y}_{|\vec{\mathtt{y}}|}) \ .$$

The string $\mathtt{y}_j$ is written over an alphabet $\Sigma_j$. In the relaxed relation $\tilde{\mathcal{R}}$, $\mathtt{y}_j$ may be a partial string over $\Sigma_j \cup \{\bot\}$. This is useful for defining the transformation of interactive oracle reductions (see Section 3.3) to interactive reductions (see Section 3.4), in Section 5. For simplicity, we assume that the number $|\vec{\mathtt{y}}|$, alphabets $\Sigma_j$, and lengths $|\mathtt{y}_j|$ of the instance oracles $\vec{\mathtt{y}} = (\mathtt{y}_1, \ldots, \mathtt{y}_{|\vec{\mathtt{y}}|})$ in every tuple $(\mathtt{i}, \mathtt{x}, \vec{\mathtt{y}}, \mathtt{w}) \in \mathcal{R}$ or $(\mathtt{i}, \mathtt{x}, \vec{\mathtt{y}}, \mathtt{w}) \in \tilde{\mathcal{R}}$ are the same.

**Relations relative to a random oracle.** A *relation relative to a random oracle*, denoted $\mathcal{R}^{\mathcal{U}}$, is a set of relations $\{\mathcal{R}^h : h \in \mathrm{supp}(\mathcal{U})\}$, where $\mathrm{supp}(\mathcal{U})$ denotes $\bigcup_{\sigma \in \mathbb{N}} \mathrm{supp}(\mathcal{U}(\sigma))$. We always name such a relation by the family $\mathcal{R}^{\mathcal{U}}$ and sample $h \xleftarrow{\$} \mathcal{U}(\sigma)$ explicitly in the experiment that uses it.

**Monotone relations.** A cheating prover may withhold parts of the strings it has committed to, i.e. it only reveals the positions that were queried by the verifier. The following (mild) condition, taken from [BMNW25], rules out that this helps the prover cheat.

**Definition 3.2** (monotone relation, [BMNW25])**.** An indexed oracle promise relation $\mathcal{R}$ is *monotone* if, whenever $(\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}) \notin \mathcal{L}(\tilde{\mathcal{R}})$ where $\mathbb{y}_j \in (\Sigma \cup \{\perp\})^{\ell_j}$ for all $j \in [|\vec{\mathbb{y}}|]$, we have $(\mathbb{i}, \mathbb{x}, (\mathbb{y}'_1, \ldots, \mathbb{y}'_{|\vec{\mathbb{y}}|})) \notin \mathcal{L}(\tilde{\mathcal{R}})$ for all $\vec{\mathbb{y}}'$ such that $\mathbb{y}'_j \preceq \mathbb{y}_j$ for all $j \in [|\vec{\mathbb{y}}|]$.

We also strengthen the notion of monotonicity to ensure that the witness is preserved under the extension of partial strings, and define *strongly monotone* relations.

**Definition 3.3** (strongly monotone relation)**.** An indexed oracle promise relation $\mathcal{R}$ is *strongly monotone* if, whenever $(\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}, \mathbb{w}) \notin \tilde{\mathcal{R}}$ where $\mathbb{y}_j \in (\Sigma \cup \{\perp\})^{\ell_j}$, we have $(\mathbb{i}, \mathbb{x}, (\mathbb{y}'_1, \ldots, \mathbb{y}'_{|\vec{\mathbb{y}}|}), \mathbb{w}) \notin \tilde{\mathcal{R}}$ for all $\vec{\mathbb{y}}'$ such that $\mathbb{y}'_j \preceq \mathbb{y}_j$ for all $j \in [|\vec{\mathbb{y}}|]$.

All relations considered in this work are strongly monotone.

## 3.2 Algebraic background

**Multilinear polynomials.** We write $\mathbb{F}^{\leq d}[X_1, \ldots, X_\mu]$ for the polynomials over $\mathbb{F}$ in $\boldsymbol{X} = (X_1, \ldots, X_\mu)$ of individual degree at most $d$. The multilinear polynomials $\mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ are uniquely determined by their evaluations on $\{0,1\}^\mu$. The *multilinear extension* of $\boldsymbol{m} \in \mathbb{F}^{2^\mu}$ is the unique $\hat{m} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ satisfying $\hat{m}(\langle i-1 \rangle) = \boldsymbol{m}[i]$ for all $i \in [2^\mu]$, where $\langle j \rangle$ denotes the $\mu$-bit representation of $j$.

**Lagrange basis polynomial.** The *Lagrange basis polynomial* $\mathrm{eq}_\mu(\boldsymbol{X}, \boldsymbol{Y}) := \prod_{i=1}^{\mu}(X_iY_i + (1-X_i)(1-Y_i))$ is multilinear and equals the equality indicator on $\{0,1\}^\mu \times \{0,1\}^\mu$. We omit the subscript when clear. Every $p \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ satisfies, for $\boldsymbol{z} \in \mathbb{F}^\mu$,

$$p(\boldsymbol{z}) = \sum_{\boldsymbol{b} \in \{0,1\}^\mu} p(\boldsymbol{b}) \cdot \mathrm{eq}(\boldsymbol{b}, \boldsymbol{z}) \ .$$

**Lemma 3.4** (Polynomial identity lemma [Sch80; Zip79; DL78])**.** *Let $f \in \mathbb{F}^{\leq d}[X_1, \ldots, X_\mu]$ be a non-zero polynomial. Then* $\Pr_{\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^\mu}[f(\boldsymbol{r}) = 0] \leq \frac{d \cdot \mu}{|\mathbb{F}|}$.

**Definition 3.5** (Linear codes)**.** A linear code $\mathcal{C}$ of message length $k$ and block length $b$ is a $k$-dimensional subspace of $\mathbb{F}^b$. Its *rate* is $\rho_{\mathcal{C}} := k/b$ and its relative Hamming *distance* is $\delta_{\mathcal{C}} := \min_{\boldsymbol{x} \neq \boldsymbol{y} \in \mathcal{C}} \delta(\boldsymbol{x}, \boldsymbol{y})$; we denote these parameters by $(\mathbb{F}, b, k, \delta_{\mathcal{C}})$. Each code has an *encoding function*, a linear bijection $\mathsf{Enc}_{\mathcal{C}} : \mathbb{F}^k \to \mathcal{C}$.

We write $\boldsymbol{c} \in \mathcal{C}$ for codewords and $\boldsymbol{w} \in (\mathbb{F} \cup \{\perp\})^b$ for (possibly partial) arbitrary words. Without loss of generality, we assume that the message length of every code considered in this paper is a power of two.

**Definition 3.6** (List decoding)**.** For a linear code $\mathcal{C}$, a word $\boldsymbol{w} \in (\mathbb{F} \cup \{\perp\})^b$, and a proximity parameter $\delta$, define $\Lambda(\mathcal{C}, \boldsymbol{w}, \delta) := \{\boldsymbol{c} \in \mathcal{C} : \boldsymbol{w} \sim_\delta \boldsymbol{c}\}$. We also define $\Lambda(\mathcal{C}, \delta) := \max_{\boldsymbol{w}} |\Lambda(\mathcal{C}, \boldsymbol{w}, \delta)|$.

**Code-based relations.** We recall the sumcheck-evaluation proximity relation (SEP) from [BMMS26b].

**Indexed oracle relation 3.7.** The *sumcheck-evaluation proximity relation* $\mathsf{SEP}[\mathcal{C}, \delta] = (\mathcal{R}, \tilde{\mathcal{R}})$ has index $(\mathcal{C}, \delta)$, where $\mathcal{C}$ has parameters $(\mathbb{F}, b, k, \delta_{\mathcal{C}})$ and $\delta \in (0,1)$. Its instance is $(f(\boldsymbol{X}, \boldsymbol{Z}), \zeta)$ where $f(\boldsymbol{X}, Z) \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_{\log k + 1}]$ and $\zeta \in \mathbb{F}$, and its witness is $\boldsymbol{m} \in \mathbb{F}^k$. Both relations require $\sum_{\boldsymbol{b} \in \{0,1\}^{\log k}} f(\boldsymbol{b}, \hat{m}(\boldsymbol{b})) = \zeta$. In addition, $\boldsymbol{w} = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m})$ in $\mathcal{R}$ and $\boldsymbol{w} \sim_\delta \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m})$ in $\tilde{\mathcal{R}}$, where $\boldsymbol{w} \in (\mathbb{F} \cup \{\perp\})^b$ is the instance oracle.

**Indexed oracle relation 3.8.** The *multilinear-evaluation proximity relation* $\mathsf{MEP}[\mathcal{C}, \delta] = (\mathcal{R}, \tilde{\mathcal{R}})$ has the same index $(\mathcal{C}, \delta)$ and witness $\boldsymbol{m} \in \mathbb{F}^k$ as $\mathsf{SEP}[\mathcal{C}, \delta]$. Its instance is $(\boldsymbol{z}, y)$, where $\boldsymbol{z} \in \mathbb{F}^{\log k}$ and $y \in \mathbb{F}$. Both relations require $\hat{m}(\boldsymbol{z}) = y$. In addition, $\boldsymbol{w} = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m})$ in $\mathcal{R}$ and $\boldsymbol{w} \sim_\delta \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m})$ in $\tilde{\mathcal{R}}$, where $\boldsymbol{w} \in (\mathbb{F} \cup \{\perp\})^b$ is the instance oracle.

**Remark 3.9** ($\mathsf{MEP}$ is a special case of $\mathsf{SEP}$)**.** Setting $f(\boldsymbol{X}, Z) := Z \cdot \mathrm{eq}(\boldsymbol{X}, \boldsymbol{z})$ and $\zeta := y$ converts an $\mathsf{MEP}[\mathcal{C}, \delta]$ claim into an equivalent $\mathsf{SEP}[\mathcal{C}, \delta]$ claim in both the strict and relaxed relations. Thus an IOPP for $\mathsf{SEP}[\mathcal{C}, \delta]$ gives an IOPP for $\mathsf{MEP}[\mathcal{C}, \delta]$ with essentially the same efficiency parameters.

### 3.3 Interactive oracle reductions (IORs)

We recall the formal definition of IORs from [BCG+19; BMNW25].

**Definition 3.10** (Interactive oracle reductions)**.** Let $\mathcal{R}$ and $\mathcal{R}'$ be indexed oracle promise relations. A (public-coin, holographic) interactive oracle reduction from $\mathcal{R}$ to $\mathcal{R}'$ is a tuple of polynomial-time algorithms $\mathsf{IOR} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ with the following syntax. The index, instance, and proof strings use alphabet $\Sigma$. The description of $\mathsf{IOR}$ specifies $\mathsf{k}$ rounds and, for each $i \in [\mathsf{k}]$, a finite nonempty challenge set $R_i$.

- The deterministic indexer $\mathbf{I}$ takes an index $\mathbb{i}$ for $\mathcal{R}$ and outputs a *short index* $\iota$, an *index oracle* $\mathbb{I}$, and a new index $\mathbb{i}'$ for $\mathcal{R}'$.[3]
- The prover $\mathbf{P}$ receives the index $\mathbb{i}$, short instance $\mathbb{x}$, instance oracles $\vec{\mathbb{y}} = (\mathbb{y}_1, \ldots, \mathbb{y}_{|\vec{\mathbb{y}}|})$, and witness $\mathbb{w}$.
- The verifier $\mathbf{V}$ receives the short index $\iota$ and short instance $\mathbb{x}$ explicitly, and oracle access to $\mathbb{I}$ and $\vec{\mathbb{y}}$.

In round $i$, $\mathbf{P}$ sends the prover message $M_i := (\Pi_i, \alpha_i)$, where $\Pi_i$ is a proof oracle and $\alpha_i$ is a plaintext (non-oracle) message; either component may be empty. The verifier $\mathbf{V}$ receives oracle access to $\Pi_i$ and explicit access to $\alpha_i$, then samples $\rho_i \xleftarrow{\$} R_i$ uniformly at random and sends it to $\mathbf{P}$. We write the full interaction transcript as $\mathsf{tr} = (M_1, \rho_1, \ldots, M_{\mathsf{k}}, \rho_{\mathsf{k}})$.

At the end of the interaction, $\mathbf{V}$ outputs a decision $b \in \{0, 1\}$ and a new instance $(\mathbb{x}', \vec{\mathbb{y}}')$. The output instance oracles are selected from the input instance oracles and proof oracles.[4] The ordered selection of output instance oracles is predetermined and does not depend on the instance or the interaction transcript. The prover outputs $(b, \mathbb{x}', \vec{\mathbb{y}}', \mathbb{w}')$. We denote the IOR interaction by

$$(b, \mathbb{x}', \vec{\mathbb{y}}', \mathbb{w}') \leftarrow \left\langle \mathbf{P}(\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}, \mathbb{w}), \mathbf{V}^{\mathbb{I}, \vec{\mathbb{y}}}(\iota, \mathbb{x}) \right\rangle \ .$$

Without loss of generality, $\mathbf{P}$ always moves first and $\mathbf{V}$ is split into two phases. In the *interaction phase*, it samples challenges and sends them to the prover. In the *query phase*, it queries the index, instance, and proof oracles; each query may depend on answers to earlier queries, including queries to other oracles. All verifier randomness, including randomness used to choose queries, is included in $\vec{\rho}$. Thus, given its explicit inputs, plaintext messages $\vec{\alpha}$, and challenges $\vec{\rho}$, the verifier is deterministic with oracle access to the index, instance, and proof oracles. We denote its output by

$$(b, \mathbb{x}', \vec{\mathbb{y}}') \leftarrow \mathbf{V}^{\mathbb{I}, \vec{\mathbb{y}}, \vec{\Pi}}(\iota, \mathbb{x}; \vec{\alpha}, \vec{\rho}) \ .$$

[3] We assume that an IOR has a single index oracle. This is without loss of generality: a fixed tuple of index oracles over the common alphabet $\Sigma$ can be concatenated into one oracle, with queries translated using the fixed lengths of the original oracles.

[4] The index oracle is not selected as an output oracle. This restriction does not limit composition: if a subsequent IOR needs to query the index oracle, we can retain the current index in the intermediate index, and let the indexer of the subsequent IOR initialize the index oracle.

**Completeness.** IOR is complete if, for every $(\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}, \mathbb{w}) \in \mathcal{R}$, running $(\iota, \mathbb{I}, \mathbb{i}') \leftarrow \mathbf{I}(\mathbb{i})$ followed by the honest interaction yields $b = 1$ and $(\mathbb{i}', \mathbb{x}', \vec{\mathbb{y}}', \mathbb{w}') \in \mathcal{R}'$ with probability 1.

**Knowledge soundness.** We recall the notion of round-by-round knowledge soundness (RBRKS) from [BCFW25], adapted to use our IOR notation. In the following, let $\mathsf{IOR} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ be an IOR from $\mathcal{R}$ to $\mathcal{R}'$ with $\mathsf{k}$ rounds, where $\mathcal{R}$ and $\mathcal{R}'$ are indexed oracle promise relations.

**Definition 3.11.** A *knowledge state function* for IOR is a (possibly inefficient) function KState that, on input a statement $(\mathbb{i}, \mathbb{x}, \mathbb{y})$, interaction transcript $\mathsf{tr}$, and *knowledge state witness* $\mathsf{w}$, outputs a bit and has the following properties.

- Empty transcript: if $\mathsf{tr} = \varnothing$ is the empty transcript, then $\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathsf{w}) = 1$ if and only if $(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{w}) \in \tilde{\mathcal{R}}$.
- Prover moves: if $\mathsf{tr}$ is a transcript where the prover is about to move and $\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathsf{w}) = 0$, then $\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, M, \mathsf{w}) = 0$ for every prover message $M = (\Pi, \alpha)$.
- Full transcript: let $\mathsf{tr} = (M_1, \rho_1, \ldots, M_{\mathsf{k}}, \rho_{\mathsf{k}})$ be a full transcript with $M_i = (\Pi_i, \alpha_i)$, and compute

$$(\iota, \mathbb{I}, \mathbb{i}') \leftarrow \mathbf{I}(\mathbb{i}) \ ,$$
$$(b, \mathbb{x}', \mathbb{y}') \leftarrow \mathbf{V}^{\mathbb{I}, \mathbb{y}, \vec{\Pi}}(\iota, \mathbb{x}; \vec{\alpha}, \vec{\rho}) \ .$$

Then $\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathsf{w}) = 1$ if and only if $b = 1$ and $(\mathbb{i}', \mathbb{x}', \mathbb{y}', \mathsf{w}) \in \tilde{\mathcal{R}}'$.

**Definition 3.12.** IOR has *round-by-round knowledge soundness errors* $(\kappa_i)_{i \in [\mathsf{k}]}$, *and extraction times* $(\mathsf{et}_i)_{i \in [\mathsf{k}]}$ if there exists a knowledge state function KState for IOR and a deterministic extractor $\mathcal{E}_{\text{rbr}}$ with the following property: for every statement $(\mathbb{i}, \mathbb{x}, \mathbb{y})$ such that $|(\mathbb{i}, \mathbb{x}, \mathbb{y})| \leq n$, round index $i \in [\mathsf{k}]$, and interaction transcript $\mathsf{tr} = (M_1, \rho_1, \ldots, M_{i-1}, \rho_{i-1}, M_i)$ where the verifier is about to move, $\mathcal{E}_{\text{rbr}}$ runs in time at most $\mathsf{et}_i$ and

$$\Pr_{\rho_i \xleftarrow{\$} R_i}\left[\exists\, \mathsf{w} : \begin{array}{c} \mathsf{KState}\big(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathcal{E}_{\text{rbr}}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \rho_i, \mathsf{w})\big) = 0 \\ \wedge\ \mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \rho_i, \mathsf{w}) = 1 \end{array}\right] \leq \kappa_i(n) \ .$$

The *total extraction time* is defined to be $\mathsf{et} := \sum_{i \in [\mathsf{k}]} \mathsf{et}_i$.

**Definition 3.13** (strongly monotone knowledge state)**.** An IOR knowledge state KState is *strongly monotone* if extending any input or proof oracle preserves its value 1, with the index, short instance, plaintext messages, challenges, and knowledge state witness fixed. Formally, whenever $\mathbb{y}'$ extends $\mathbb{y}$ component-wise and $\mathsf{tr}'$ is obtained from a transcript prefix $\mathsf{tr}$ by replacing each $M_i = (\Pi_i, \alpha_i)$ with $M_i' = (\Pi_i', \alpha_i)$ where $\Pi_i'$ extends $\Pi_i$, we have, for every $\mathbb{i}, \mathbb{x}, \mathsf{w}$,

$$\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathsf{w}) = 1 \quad \Longrightarrow \quad \mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}', \mathsf{tr}', \mathsf{w}) = 1 \ .$$

All the IOR knowledge state functions considered in this work are strongly monotone.

**Remark 3.14** (IOPPs and IOPs)**.** An Interactive Oracle Proof of Proximity (IOPP) for an indexed oracle promise relation $\mathcal{R}$ is an IOR from $\mathcal{R}$ to the trivial relation $\mathcal{R}_{\mathsf{true}} := \{(\mathbb{i} := \bot, \mathbb{x} := \mathsf{accept}, \mathbb{y} := \bot, \mathbb{w} := \bot)\}$. An Interactive Oracle Proof (IOP) is an IOPP whose input relation $\mathcal{R}$ is an ordinary indexed relation: its tuples have no instance oracles and its strict and relaxed relations are identical.

**Efficiency measures.** We measure prover and verifier time, oracle lengths, total plaintext communication $\sum_{j=1}^{\mathsf{k}} |\alpha_j|$, and query complexity (the number of oracle locations read by the verifier). The round complexity is $\mathsf{k}$. These measures may depend on the short index $\iota$.

### 3.4 Interactive reductions in the ROM

Interactive reductions (IRs) are public-coin interactive protocols that reduce membership in one indexed promise relation to membership in another. Unlike IORs, they do not give the verifier oracle access to prover messages, and all prover communication is sent explicitly. The protocol algorithms and adversaries operate in the random oracle model: a single random oracle $h \xleftarrow{\$} \mathcal{U}(\sigma)$ is sampled at the beginning of each experiment, and all oracle access is to this same $h$. The round-by-round knowledge extractors defined in Definition 3.17 receive the random-oracle traces of the adversary, indexer, and verifier and have no random-oracle access.

**Definition 3.15** (Interactive reduction)**.** Let $\mathcal{R}_1^{\mathcal{U}}$ and $\mathcal{R}_2^{\mathcal{U}}$ be indexed promise relations relative to a random oracle. A *interactive reduction* from $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_2^{\mathcal{U}}$ is a tuple of polynomial-time oracle algorithms

$$\mathsf{IR} = (\mathbf{I}_{\mathsf{IR}}, \mathbf{P}_{\mathsf{IR}}, \mathbf{V}_{\mathsf{IR}})$$

with the following syntax. For every round $i \in [\mathsf{k}_{\mathsf{IR}}]$, the description of $\mathsf{IR}$ specifies a polynomial upper bound on the prover-message length and a finite nonempty challenge set $R_i$.

- $\mathbf{I}_{\mathsf{IR}}^h(\mathbb{i})$ is deterministic given $h$ and outputs an index $\mathbb{i}'$ for $\mathcal{R}_2$, a prover key $\mathsf{pk}$, and a verifier key $\mathsf{vk}$.
- $\mathbf{P}_{\mathsf{IR}}^h(\mathsf{pk}, \mathbb{x}, \mathbb{w})$ interacts with $\mathbf{V}_{\mathsf{IR}}^h(\mathsf{vk}, \mathbb{x})$ for $\mathsf{k}_{\mathsf{IR}}$ public-coin rounds. In round $i$, $\mathbf{P}_{\mathsf{IR}}$ sends a message $\alpha_i$ and $\mathbf{V}_{\mathsf{IR}}$ samples $\rho_i \xleftarrow{\$} R_i$ uniformly at random and sends it to $\mathbf{P}_{\mathsf{IR}}$. At the end of the interaction, $\mathbf{V}_{\mathsf{IR}}$ outputs a decision $b \in \{0,1\}$ and a new instance $\mathbb{x}'$. $\mathbf{P}_{\mathsf{IR}}$ outputs a new witness $\mathbb{w}'$. We denote the resulting execution by
$$(b, \mathbb{x}', \mathbb{w}') \xleftarrow{\mathsf{tr}} \left\langle \mathbf{P}_{\mathsf{IR}}^h(\mathsf{pk}, \mathbb{x}, \mathbb{w}), \mathbf{V}_{\mathsf{IR}}^h(\mathsf{vk}, \mathbb{x}) \right\rangle .$$
The interaction transcript $\mathsf{tr}$ consists of all prover messages and verifier challenges. We also use $(b, \mathbb{x}') \leftarrow \mathbf{V}_{\mathsf{IR}}^h(\mathsf{vk}, \mathbb{x}; \vec{\alpha}, \vec{\rho})$ to denote the output of $\mathbf{V}_{\mathsf{IR}}$ given the prover messages $\vec{\alpha}$ and verifier challenges $\vec{\rho}$. For a transcript prefix $\mathsf{tr}$ that is empty or ends with a challenge, $\mathbf{V}_{\mathsf{IR}}^h(\mathsf{vk}, \mathbb{x}; \mathsf{tr})$ denotes the execution of $\mathbf{V}_{\mathsf{IR}}$ that takes its challenges from $\mathsf{tr}$ and runs until it has sent the last challenge in $\mathsf{tr}$, computing its output as well when $\mathsf{tr}$ is full. Since $\mathbf{V}_{\mathsf{IR}}$ is deterministic given $h$ and its challenges, its trace on a prefix is an initial segment of its trace on every extension of the prefix.

An interactive reduction satisfies the following properties.

**Completeness.** $\mathsf{IR}$ is complete if, for all $\sigma \in \mathbb{N}$, random oracle $h \in \mathrm{supp}(\mathcal{U}(\sigma))$, and $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_1^h$, we have

$$\Pr\left[ b = 1 \wedge (\mathbb{i}', \mathbb{x}', \mathbb{w}') \in \mathcal{R}_2^h \;\middle|\; \begin{array}{l} (\mathbb{i}', \mathsf{pk}, \mathsf{vk}) \leftarrow \mathbf{I}_{\mathsf{IR}}^h(\mathbb{i}) \\ (b, \mathbb{x}', \mathbb{w}') \leftarrow \left\langle \mathbf{P}_{\mathsf{IR}}^h(\mathsf{pk}, \mathbb{x}, \mathbb{w}), \mathbf{V}_{\mathsf{IR}}^h(\mathsf{vk}, \mathbb{x}) \right\rangle \end{array} \right] = 1 \ .$$

**Round-by-round knowledge soundness.** We define a notion of round-by-round knowledge soundness for interactive reductions in the ROM following the corresponding definitions for IORs (Definitions 3.11 and 3.12). In the following, let $\mathcal{R}_1^{\mathcal{U}}$ and $\mathcal{R}_2^{\mathcal{U}}$ be indexed promise relations, and $\mathsf{IR} = (\mathbf{I}_{\mathsf{IR}}, \mathbf{P}_{\mathsf{IR}}, \mathbf{V}_{\mathsf{IR}})$ be an IR from $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_2^{\mathcal{U}}$ with $\mathsf{k}_{\mathsf{IR}}$ rounds.

**Definition 3.16.** Let $\mathsf{IR} = (\mathbf{I}_{\mathsf{IR}}, \mathbf{P}_{\mathsf{IR}}, \mathbf{V}_{\mathsf{IR}})$ be an IR from a relation $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_2^{\mathcal{U}}$. Let $h \xleftarrow{\$} \mathcal{U}(\sigma)$ be a random oracle. A *knowledge state function* for $\mathsf{IR}$ is a (possibly inefficient) function $\mathsf{KState}^h$ that, on input a statement $(\mathbb{i}, \mathbb{x})$, interaction transcript $\mathsf{tr}$, and *knowledge state witness* $\mathsf{w}$, outputs a bit and has the following properties.

- Empty transcript: if $\mathsf{tr} = \varnothing$ is the empty transcript, then $\mathsf{KState}^h(\mathbb{i}, \mathbb{x}, \mathsf{tr}, \mathsf{w}) = 1$ if and only if $(\mathbb{i}, \mathbb{x}, \mathsf{w}) \in \tilde{\mathcal{R}}_1^h$.

- Prover moves: if $\mathsf{tr}$ is a transcript where the prover is about to move and $\mathsf{KState}^h(\mathbb{i}, \mathbb{x}, \mathsf{tr}, \mathsf{w}) = 0$, then $\mathsf{KState}^h(\mathbb{i}, \mathbb{x}, \mathsf{tr} \,\|\, \alpha, \mathsf{w}) = 0$ for every prover message $\alpha$.
- Full transcript: if $\mathsf{tr} = (\alpha_1, \rho_1, \ldots, \alpha_{\mathsf{k_{IR}}}, \rho_{\mathsf{k_{IR}}})$ is a full transcript, then $\mathsf{KState}^h(\mathbb{i}, \mathbb{x}, \mathsf{tr}, \mathsf{w}) = 1$ if and only if $\mathbf{V}^h_{\mathsf{IR}}(\mathsf{vk}, \mathbb{x}; \vec{\alpha}, \vec{\rho})$ outputs $(b, \mathbb{x}')$ such that $b = 1$ and $(\mathbb{i}', \mathbb{x}', \mathsf{w}) \in \tilde{\mathcal{R}}_2^h$, where $(\mathbb{i}', \mathsf{pk}, \mathsf{vk}) \leftarrow \mathbf{I}^h_{\mathsf{IR}}(\mathbb{i})$.

**Definition 3.17.** $\mathsf{IR}$ has *round-by-round knowledge soundness errors* $(\kappa_i(\sigma, n, q))_{i \in [\mathsf{k_{IR}}]}$, *and extraction times* $(\mathsf{et}_i(\sigma, n, q))_{i \in [\mathsf{k_{IR}}]}$ if there exists a knowledge state function $\mathsf{KState}^h$ for $\mathsf{IR}$ and a deterministic extractor $\mathcal{E}_{\text{rbr}}$ that makes no random-oracle queries and runs in time at most $\mathsf{et}_i(\sigma, n, q)$ in round $i \in [\mathsf{k_{IR}}]$ such that the following holds for every fixed round $i \in [\mathsf{k_{IR}}]$ and every stateful $q$-query ROM adversary $\mathcal{A}$:

$$\Pr\left[\begin{array}{l|l} & h \xleftarrow{\$} \mathcal{U}(\sigma) \\ & (\mathbb{i}, \mathbb{x}, \mathsf{tr}) \xleftarrow{\mathsf{qtr}_0} \mathcal{A}^h(1^\sigma) \\ & \rho_i \xleftarrow{\$} R_i \\ |(\mathbb{i}, \mathbb{x})| \le n & \mathsf{w}' \xleftarrow{\mathsf{qtr}_1} \mathcal{A}^h(\rho_i) \\ \wedge\ \mathsf{KState}^h(\mathbb{i}, \mathbb{x}, \mathsf{tr}, \mathsf{w}) = 0 & (\mathbb{i}', \mathsf{pk}, \mathsf{vk}) \xleftarrow{\mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}}} \mathbf{I}^h_{\mathsf{IR}}(\mathbb{i}) \\ \wedge\ \mathsf{KState}^h(\mathbb{i}, \mathbb{x}, \mathsf{tr} \,\|\, \rho_i, \mathsf{w}') = 1 & \mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}} \leftarrow \text{trace of } \mathbf{V}^h_{\mathsf{IR}}(\mathsf{vk}, \mathbb{x}; \mathsf{tr} \,\|\, \rho_i) \\ & \mathsf{w} \leftarrow \mathcal{E}_{\text{rbr}}(\mathbb{i}, \mathbb{x}, \mathsf{qtr}_0, \mathsf{qtr}_1, \\ & \qquad \mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}}, \mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}}, \mathsf{tr} \,\|\, \rho_i, \mathsf{w}') \end{array}\right] \le \kappa_i(\sigma, n, q) \ .$$

In the experiment above, $\mathcal{A}$ first outputs $(\mathbb{i}, \mathbb{x}, \mathsf{tr})$, where $\mathsf{tr} = (\alpha_1, \rho_1, \ldots, \alpha_{i-1}, \rho_{i-1}, \alpha_i)$ is a transcript prefix ending immediately before the verifier's $i$th challenge, and $\mathbf{V}_{\mathsf{IR}}$ is run on the prefix $\mathsf{tr} \,\|\, \rho_i$ as in Definition 3.15. The challenger then samples a fresh challenge $\rho_i$ and resumes the same adversary on input $\rho_i$, with its internal state retained, to obtain $\mathsf{w}'$. The *total extraction time* is defined to be $\mathsf{et} := \sum_{i \in [\mathsf{k_{IR}}]} \mathsf{et}_i(\sigma, n, q)$.

**Remark 3.18** (Extraction time and trace lengths)**.** The extractor of Definition 3.17 reads the random-oracle traces of the adversary, the indexer, and the verifier, so its running time also depends on the bit lengths of these traces, which $(\sigma, n, q)$ do not bound. For brevity, we omit this contribution from every extraction time in this paper.

In Lemma C.4, we show that round-by-round knowledge soundness implies state-restoration knowledge soundness for IRs. In Lemma C.5, we show that round-by-round knowledge soundness is preserved under composition of IRs.

# 4 Vector Commit-and-Reduce (VCR) schemes

Vector Commit-and-Reduce (VCR) schemes generalize vector commitment schemes [CF13] in two ways:
- The commitment protocol and the opening reduction can be interactive.
- Instead of the verifier simply accepting or rejecting the opening claim, the opening reduction outputs a claim about an indexed promise relation.

Our definition is modeled after the definition of interactive oracle commitments by [BFRW26], but tailored to vector commitments in the ROM, without the use of additional oracles.

In Section 4.1, we give the formal definition of VCR schemes, ending on a few remarks about the intuition behind the definition. In Section 4.2, we describe the VCR scheme associated with a VC scheme and show how VCR extractability, together with perfect knowledge soundness of the opening reduction, implies VC extractability as defined in [CY24, Section 18] for Merkle trees. In Section 4.3, we recall the definition of Merkle trees as a VC scheme, and state the properties needed for the construction of the Canopy VCR scheme in Section 6.

## 4.1 Formal definition

**Definition 4.1** (VCR scheme)**.** A *vector commit-and-reduce* (VCR) scheme with random-oracle output length $\sigma$, alphabet $\Sigma$, and message length $\ell$, denoted $\mathsf{VCR}[\sigma, \Sigma, \ell]$, consists of a public-coin interactive protocol $\mathsf{Com}$, an IR $\mathsf{Rdx}$, an indexed relation $\mathcal{R}^{\mathcal{U}}_{\mathsf{bad}}$, a commitment relation $\mathcal{R}^{\mathcal{U}}_{\mathsf{com}}$ and an output relation $\mathcal{R}^{\mathcal{U}}_{\mathsf{out}}$ (the latter two are indexed promise relations):

$$\mathsf{VCR}[\sigma, \Sigma, \ell] := \left(\mathsf{Com}, \mathsf{Rdx}, \mathcal{R}^{\mathcal{U}}_{\mathsf{bad}}, \mathcal{R}^{\mathcal{U}}_{\mathsf{com}}, \mathcal{R}^{\mathcal{U}}_{\mathsf{out}}\right) \quad .$$

The role of each of the 3 relations is motivated in Remark 4.6 after the detailed definition. The bad relation $\mathcal{R}^{\mathcal{U}}_{\mathsf{bad}}$ is an ordinary relation, with no separate strict and relaxed versions. In the following, let $h \xleftarrow{\$} \mathcal{U}(\sigma)$ be a random oracle.

**Commitment protocol.** The commitment protocol $\mathsf{Com}$ consists of an indexer $\mathsf{Com}.\mathbf{I}$ and an interactive protocol between a prover $\mathsf{Com}.\mathbf{P}$ and a verifier $\mathsf{Com}.\mathbf{V}$:

- $\mathsf{Com}.\mathbf{I}^h(1^\sigma)$ is a deterministic function that outputs a prover key $\mathsf{pk}$ and a verifier key $\mathsf{vk}$.
- $\mathsf{Com}.\mathbf{P}^h(\mathsf{pk}, \boldsymbol{m})$ and $\mathsf{Com}.\mathbf{V}^h(\mathsf{vk})$ engage in a public-coin interaction, where $\boldsymbol{m} \in \Sigma^\ell$ is the message to be committed to. In the $i$-th round, $\mathsf{Com}.\mathbf{P}$ sends a message $\alpha_i$ and $\mathsf{Com}.\mathbf{V}$ samples a challenge $\rho_i \xleftarrow{\$} R_i$ uniformly at random from the finite nonempty challenge set $R_i$ specified by the protocol and sends it to $\mathsf{Com}.\mathbf{P}$. We refer to the interaction transcript as the commitment $\mathsf{cm} := (\vec{\alpha}, \vec{\rho})$. At the end of the interaction, $\mathsf{Com}.\mathbf{P}$ outputs as its witness the committed message $\boldsymbol{m}$ and a trapdoor $\mathsf{td}$ that can be used to open the commitment efficiently. We denote the honest commitment interaction by $(\mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \leftarrow \mathsf{Com}^h(\mathsf{pk}, \mathsf{vk}, \boldsymbol{m})$.

Without loss of generality for completeness and extractability, we assume that $\mathsf{Com}.\mathbf{P}$ is deterministic given its inputs, the random oracle, and $\mathsf{Com}.\mathbf{V}$'s challenges.

**Query relation.** Before defining $\mathsf{Rdx}$, we define the query relation $\mathsf{Query}[\mathcal{R}_{\mathsf{com}}]^{\mathcal{U}}$, which is the input relation of $\mathsf{Rdx}$. We write $\mathbb{i}_{\mathsf{com}} := (\mathsf{pk}, \mathsf{vk})$ for the pair of keys output by $\mathsf{Com}.\mathbf{I}$, which is the index of $\mathcal{R}^{\mathcal{U}}_{\mathsf{com}}$, of $\mathcal{R}^{\mathcal{U}}_{\mathsf{bad}}$, and of the query relation, and we write $\mathsf{Com}^h(\mathbb{i}_{\mathsf{com}}, \boldsymbol{m})$ for $\mathsf{Com}^h(\mathsf{pk}, \mathsf{vk}, \boldsymbol{m})$. The query relation is

$$\mathsf{Query}[\mathcal{R}_{\mathsf{com}}]^h := \left\{ (\mathbb{i}_{\mathsf{com}}, (\mathsf{cm}, Q, \boldsymbol{a}), (\boldsymbol{m}, \mathsf{td})) : \begin{array}{l} (\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \mathcal{R}^h_{\mathsf{com}} \\ \wedge\ \boldsymbol{m}[Q] = \boldsymbol{a} \end{array} \right\} \tag{2}$$

The relaxed relation $\widetilde{\mathsf{Query}}[\tilde{\mathcal{R}}_{\mathsf{com}}]^h$ is defined analogously, replacing $\mathcal{R}^h_{\mathsf{com}}$ by $\tilde{\mathcal{R}}^h_{\mathsf{com}}$.

**Opening reduction.** The opening reduction $\mathsf{Rdx}$ is an IR from $\mathsf{Query}[\mathcal{R}_{\mathsf{com}}]^{\mathcal{U}}$ to $\mathcal{R}^{\mathcal{U}}_{\mathsf{out}}$ consisting of an indexer $\mathsf{Rdx}.\mathbf{I}$, a prover $\mathsf{Rdx}.\mathbf{P}$, and a verifier $\mathsf{Rdx}.\mathbf{V}$. The components have the following syntax.

- $\mathsf{Rdx}.\mathbf{I}^h(\mathbb{i})$, on input $\mathbb{i} = \mathbb{i}_{\mathsf{com}}$, outputs an index $\mathbb{i}_{\mathsf{out}}$ for $\mathcal{R}^{\mathcal{U}}_{\mathsf{out}}$ and the prover and verifier keys $\mathsf{pk}_{\mathsf{Rdx}}$ and $\mathsf{vk}_{\mathsf{Rdx}}$ for the IR. On any other index it outputs $(\bot, \bot, \bot)$.
- $\mathsf{Rdx}.\mathbf{P}^h(\mathsf{pk}_{\mathsf{Rdx}}, (\mathsf{cm}, Q, \boldsymbol{a}), (\boldsymbol{m}, \mathsf{td}))$ and $\mathsf{Rdx}.\mathbf{V}^h(\mathsf{vk}_{\mathsf{Rdx}}, (\mathsf{cm}, Q, \boldsymbol{a}))$ engage in a public-coin interaction, where $Q \subseteq [\ell]$ is a query set and $\boldsymbol{a} \in \Sigma^Q$ is a claimed restriction of the committed message to the coordinates in $Q$. $\mathsf{Rdx}.\mathbf{V}$ outputs a decision $b$ and an instance $\mathbb{x}_{\mathsf{out}}$, while $\mathsf{Rdx}.\mathbf{P}$ outputs a witness $\mathbb{w}_{\mathsf{out}}$.

We use the following shorthand to denote the result of the interaction:

$$\begin{aligned}&\mathsf{Rdx}^h((\mathsf{pk}_{\mathsf{Rdx}}, \mathsf{vk}_{\mathsf{Rdx}}), (\mathsf{cm}, Q, \boldsymbol{a}), (\boldsymbol{m}, \mathsf{td}))\\ &\quad := \left\langle \mathsf{Rdx}.\mathbf{P}^h(\mathsf{pk}_{\mathsf{Rdx}}, (\mathsf{cm}, Q, \boldsymbol{a}), (\boldsymbol{m}, \mathsf{td})), \mathsf{Rdx}.\mathbf{V}^h(\mathsf{vk}_{\mathsf{Rdx}}, (\mathsf{cm}, Q, \boldsymbol{a})) \right\rangle \ .\end{aligned}$$

Thus, we have

$$(b, \mathbb{x}_{\mathsf{out}}, \mathbb{w}_{\mathsf{out}}) \leftarrow \mathsf{Rdx}^h((\mathsf{pk}_{\mathsf{Rdx}}, \mathsf{vk}_{\mathsf{Rdx}}), (\mathsf{cm}, Q, \boldsymbol{a}), (\boldsymbol{m}, \mathsf{td})) \ .$$

$(\mathbb{i}_{\mathsf{out}}, \mathbb{x}_{\mathsf{out}}, \mathbb{w}_{\mathsf{out}})$ is a purported index-instance-witness tuple for $\mathcal{R}^h_{\mathsf{out}}$.

We require a VCR scheme to satisfy the following properties, each stated for a scheme $\mathsf{VCR}[\sigma, \Sigma, \ell]$.

**Definition 4.2** (Completeness of $\mathsf{Com}$)**.** The commitment protocol $\mathsf{Com}$ of $\mathsf{VCR}[\sigma, \Sigma, \ell]$ is complete if for every message $\boldsymbol{m} \in \Sigma^\ell$, we have

$$\Pr\left[\begin{array}{l|l} (\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \mathcal{R}^h_{\mathsf{com}} & h \xleftarrow{\$} \mathcal{U}(\sigma) \\ \wedge \ (\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, \mathsf{td}) \notin \mathcal{R}^h_{\mathsf{bad}} & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Com}.\mathbf{I}^h(1^\sigma) \\ & (\mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \leftarrow \mathsf{Com}^h(\mathsf{pk}, \mathsf{vk}, \boldsymbol{m}) \end{array}\right] = 1 \ .$$

**Definition 4.3** (Extractability of $\mathsf{Com}$)**.** The commitment protocol $\mathsf{Com}$ of $\mathsf{VCR}[\sigma, \Sigma, \ell]$ has extractability error $\kappa(\sigma, \ell, q)$ with extraction time $\mathsf{et}(\sigma, \ell, q)$ if there exists a deterministic extractor $\mathsf{Com}.\mathbf{E}$ that runs in time at most $\mathsf{et}(\sigma, \ell, q)$ such that the following holds for every $q \in \mathbb{N}$ and stateful $q$-query ROM adversary $\mathcal{A}$:

$$\Pr\left[\begin{array}{l|l} (\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \tilde{\mathcal{R}}^h_{\mathsf{com}} & h \xleftarrow{\$} \mathcal{U}(\sigma) \\ \wedge \ \left( \begin{array}{ll} & (\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}', \mathsf{td}')) \notin \tilde{\mathcal{R}}^h_{\mathsf{com}} \\ \vee & \left( (\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, \mathsf{td}') \notin \mathcal{R}^h_{\mathsf{bad}} \wedge \boldsymbol{m} \not\preceq \boldsymbol{m}' \right) \end{array} \right) & \begin{array}{l} (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Com}.\mathbf{I}^h(1^\sigma) \\ \mathsf{cm} \xleftarrow{\mathsf{qtr}} \mathcal{A}^h(1^\sigma, \mathsf{pk}) \\ (\boldsymbol{m}', \mathsf{td}') \leftarrow \mathsf{Com}.\mathbf{E}(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, \mathsf{qtr}) \\ (\boldsymbol{m}, \mathsf{td}) \leftarrow \mathcal{A}^h(1^\sigma, \mathsf{pk}) \end{array} \end{array}\right] \tag{3}$$

is at most $\kappa(\sigma, \ell, q)$. The adversary outputs the entire commitment in its first stage and retains its state for the second stage, with at most $q$ random-oracle queries across both stages. At a high level, the definition says that whenever the adversary's witness is valid with respect to the commitment $\mathsf{cm}$ (that means $(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \tilde{\mathcal{R}}^h_{\mathsf{com}}$), successful extraction requires both of the following conditions to hold: (a) the extractor's witness is valid with respect to $\mathsf{cm}$, and (b) $\boldsymbol{m} \preceq \boldsymbol{m}'$ if the commitment and extracted trapdoor lie outside $\mathcal{R}_{\mathsf{bad}}$. When the extracted trapdoor lies in $\mathcal{R}_{\mathsf{bad}}$, no extension requirement is imposed on $\boldsymbol{m}$.

**Definition 4.4** (round-by-round sparsity of $\mathcal{R}_{\mathsf{bad}}$)**.** Let $\mathsf{k}_{\mathsf{com}}$ be the number of rounds of the commitment protocol $\mathsf{Com}$ of $\mathsf{VCR}[\sigma, \Sigma, \ell]$. The bad relation $\mathcal{R}^{\mathcal{U}}_{\mathsf{bad}}$ of $\mathsf{VCR}[\sigma, \Sigma, \ell]$ has *round-by-round sparsity errors* $(\kappa_{\mathsf{sp},j}(\sigma, \ell, q))_{j \in [\mathsf{k}_{\mathsf{com}}]}$ if there exists a (possibly inefficient) predicate $\mathsf{B}^h(\mathbb{i}_{\mathsf{com}}, \mathsf{tr}, \boldsymbol{m}, \mathsf{td})$ on commitment transcript prefixes with the following properties, for every random oracle $h$, where $\mathbb{i}_{\mathsf{com}} = (\mathsf{pk}, \mathsf{vk})$ is the pair of keys output by $\mathsf{Com}.\mathbf{I}^h(1^\sigma)$.

- Empty transcript: $\mathsf{B}^h(\mathbb{i}_{\mathsf{com}}, \varnothing, \boldsymbol{m}, \mathsf{td}) = 0$ for every $(\boldsymbol{m}, \mathsf{td})$.
- Prover moves: if $\mathsf{tr}$ ends immediately before a prover message, then for every $(\boldsymbol{m}, \mathsf{td})$ and every next prover message $\alpha$,
$$\mathsf{B}^h(\mathbb{i}_{\mathsf{com}}, \mathsf{tr}, \boldsymbol{m}, \mathsf{td}) = 0 \implies \mathsf{B}^h(\mathbb{i}_{\mathsf{com}}, \mathsf{tr} \,\|\, \alpha, \boldsymbol{m}, \mathsf{td}) = 0 \ .$$
- Full transcript: for a full commitment transcript $\mathsf{cm}$,
$$\mathsf{B}^h(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, \boldsymbol{m}, \mathsf{td}) = 1 \iff \left\{ \begin{array}{l} (\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \tilde{\mathcal{R}}^h_{\mathsf{com}} \\ \wedge \ (\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, \mathsf{td}) \in \mathcal{R}^h_{\mathsf{bad}} \end{array} \right\} \ .$$

Moreover, for every $q \in \mathbb{N}$, round $j \in [\mathsf{k}_{\mathsf{com}}]$, and stateful $q$-query ROM adversary $\mathcal{A}$, we have

$$\Pr \left[ \begin{array}{l} \\ \mathsf{B}^h(\mathbb{i}_{\mathsf{com}}, \mathsf{tr}, \boldsymbol{m}, \mathsf{td}) = 0 \\ \wedge \ \mathsf{B}^h(\mathbb{i}_{\mathsf{com}}, \mathsf{tr} \,\|\, \rho_j, \boldsymbol{m}, \mathsf{td}) = 1 \\ \\ \end{array} \middle| \begin{array}{l} h \xleftarrow{\$} \mathcal{U}(\sigma) \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Com}.\mathbf{I}^h(1^\sigma) \\ \mathsf{tr} \leftarrow \mathcal{A}^h(1^\sigma, \mathsf{pk}) \\ \rho_j \xleftarrow{\$} R_j \\ (\boldsymbol{m}, \mathsf{td}) \leftarrow \mathcal{A}^h(\rho_j) \end{array} \right] \leq \kappa_{\mathsf{sp},j}(\sigma, \ell, q) \ ,$$

The same witness is used to evaluate $\mathsf{B}^h$ before and after the challenge, but it may be chosen after the challenge is revealed. Round-by-round sparsity implies that a $q$-query ROM adversary interacting with $\mathsf{Com}.\mathbf{V}$ outputs a valid witness whose trapdoor lies in $\mathcal{R}_{\mathsf{bad}}$ with probability at most *the total sparsity error* $\kappa_{\mathsf{sp}}(\sigma, \ell, q) = \sum_{j=1}^{\mathsf{k}_{\mathsf{com}}} \kappa_{\mathsf{sp},j}(\sigma, \ell, q)$.

**Remark 4.5** (Definitions only concern the commitment protocol)**.** The definitions of completeness, extractability, and round-by-round sparsity in Section 4 only concern the commitment protocol $\mathsf{Com}$. Since $\mathsf{Rdx}$ is an IR, its completeness and soundness are captured by the definition of IRs in Definition 3.15. This is consistent with the definition of extraction for Merkle commitment schemes in [CY24, Section 18], where the extractor only sees the commitment and the random oracle trace of the adversary, and not the explicit proof of an opening claim with respect to a specific $Q$ and $\boldsymbol{a}$.

**Remark 4.6** (Intuition for relations in VCR schemes)**.** We motivate the roles of the three relations in the VCR definition.

1. $\mathcal{R}_{\mathsf{out}}$ is the output relation of the IR $\mathsf{Rdx}$: instead of outputting a proof of point queries like a vector commitment scheme, a VCR scheme reduces a point query claim to a $\mathcal{R}_{\mathsf{out}}$ claim.

2. $\mathcal{R}_{\mathsf{com}}$ is an output relation for $\mathsf{Com}$ that represents commitment transcripts and trapdoors that are accepted as "valid" by $\mathsf{Rdx}.\mathbf{V}$. In particular, if $(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \mathcal{R}^h_{\mathsf{com}}$, then for every $Q \subseteq [\ell]$ the honest opening reduction
$$(b, \mathbb{x}_{\mathsf{out}}, \mathbb{w}_{\mathsf{out}}) \leftarrow \mathsf{Rdx}^h((\mathsf{pk}_{\mathsf{Rdx}}, \mathsf{vk}_{\mathsf{Rdx}}), (\mathsf{cm}, Q, \boldsymbol{m}[Q]), (\boldsymbol{m}, \mathsf{td}))$$
satisfies $b = 1$ and $(\mathbb{i}_{\mathsf{out}}, \mathbb{x}_{\mathsf{out}}, \mathbb{w}_{\mathsf{out}}) \in \mathcal{R}^h_{\mathsf{out}}$ with probability one by completeness, where
$$(\mathbb{i}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{Rdx}}, \mathsf{vk}_{\mathsf{Rdx}}) \leftarrow \mathsf{Rdx}.\mathbf{I}^h(\mathbb{i}_{\mathsf{com}}) \ .$$

3. $\mathcal{R}_{\mathsf{bad}}$ is the relation of all commitment transcripts that fail to provide binding for a single committed message due to bad challenges sent by $\mathsf{Com}.\mathbf{V}$. So if $\mathsf{td}'$ is the extracted trapdoor, then $(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, \mathsf{td}') \notin \mathcal{R}^h_{\mathsf{bad}}$ means that $\mathsf{Rdx}.\mathbf{P}$ cannot find two incompatible messages ($\boldsymbol{m}[i] \neq \boldsymbol{m}'[i]$ for some $i$ and both are non-bot) such that both $(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td}'))$ and $(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}', \mathsf{td}'))$ are valid (except with negligible probability). Separately, Definition 4.4 requires that $\mathcal{R}_{\mathsf{bad}}$ is round-by-round sparse under fresh verifier challenges, even after adversarially chosen transcript prefixes. This implies that the probability that an adversary's interaction with $\mathsf{Com}.\mathbf{V}$ outputs a tuple in $\tilde{\mathcal{R}}_{\mathsf{com}}$ whose commitment and trapdoor are in $\mathcal{R}_{\mathsf{bad}}$ is negligible.

**Remark 4.7** (Extractability versus round-by-round sparsity)**.** Extractability (Definition 4.3) and round-by-round sparsity (Definition 4.4) serve two different purposes. In particular, they differ in how the commitment is chosen: in the extractability definition, the entire commitment transcript is chosen by the adversary, while in the round-by-round sparsity definition, only a prefix of the commitment transcript is chosen by the adversary, and the next challenge is sampled uniformly at random. Thus, the extractability adversary can choose a $(\mathsf{cm}, \mathsf{td})$ that lies in $\mathcal{R}_{\mathsf{bad}}$, and we need to obtain an extractor that can succeed despite it (for example by outputting a $(\mathsf{cm}, \mathsf{td}')$ that also lies in $\mathcal{R}_{\mathsf{bad}}$). We need to consider such adversaries when we prove the round-by-round knowledge soundness of our transformation in Section 5. The extractability error contributes to the RBRKS errors of the rounds that occur after the commitment protocol is finished. The round-by-round sparsity error, on the other hand, determines the RBRKS error of the rounds of the commitment protocol itself.

Finally, we also discuss an additional property that a VCR scheme may or may not satisfy. Looking ahead, the VCR scheme used to instantiate the index oracle by the IR indexer in Fig. 2 must satisfy this property.

**Definition 4.8** (Honest commitment binding)**.** We say $\mathsf{VCR}[\sigma, \Sigma, \ell]$ has *honest commitment binding error* $\kappa_{\mathsf{hon}}(\sigma, \ell, q)$ if for every verifier challenge vector $\vec{\rho}$, query bound $q \in \mathbb{N}$, and stateful $q$-query ROM adversary $\mathcal{A}$:

$$\Pr\left[\begin{array}{l|l} (\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \tilde{\mathcal{R}}^h_{\mathsf{com}} & h \xleftarrow{\$} \mathcal{U}(\sigma) \\ \wedge\ \boldsymbol{m} \not\preceq \boldsymbol{m}_{\mathsf{hon}} & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Com}.\mathbf{I}^h(1^\sigma) \\ & \boldsymbol{m}_{\mathsf{hon}} \leftarrow \mathcal{A}^h(1^\sigma, \mathsf{pk}, \vec{\rho}) \\ & \mathsf{cm} \leftarrow \mathsf{HCom}^h(\mathsf{pk}, \boldsymbol{m}_{\mathsf{hon}}; \vec{\rho}) \\ & (\boldsymbol{m}, \mathsf{td}) \leftarrow \mathcal{A}^h(\mathsf{cm}) \end{array}\right] \leq \kappa_{\mathsf{hon}}(\sigma, \ell, q) \ . \tag{4}$$

Here $\mathsf{HCom}^h(\mathsf{pk}, \boldsymbol{m}_{\mathsf{hon}}; \vec{\rho})$ simulates the honest $\mathsf{Com}.\mathbf{P}$ using the given verifier challenges $\vec{\rho}$ and outputs the commitment transcript. The adversary chooses a full message $\boldsymbol{m}_{\mathsf{hon}} \in \Sigma^\ell$ in its first stage, retains its state for the second stage, and makes at most $q$ random-oracle queries across both stages.

## 4.2 Vector commitment schemes as VCR schemes

[CY24, Section 18] provides formal definitions for Merkle commitment schemes. Here, we generalize their definition to arbitrary vector commitment (VC) schemes.

**Definition 4.9.** A VC scheme $\mathsf{VC}[\sigma, \Sigma, \ell]$ with random-oracle output length $\sigma$, alphabet $\Sigma$, and message length $\ell$ consists of the following deterministic algorithms, which receive query access to a random oracle $h \xleftarrow{\$} \mathcal{U}(\sigma)$.

- $\mathsf{VC.Commit}^h(\boldsymbol{m})$: takes as input a message $\boldsymbol{m} \in \Sigma^\ell$, and outputs a commitment $\mathsf{cm}$ and a trapdoor $\mathsf{td}$.
- $\mathsf{VC.Open}^h(\mathsf{td}, Q)$: outputs an opening proof $\mathsf{pf}$ for a query index set $Q \subseteq [\ell]$.
- $\mathsf{VC.Check}^h(\mathsf{cm}, Q, \boldsymbol{a}, \mathsf{pf})$: outputs a bit $b$ indicating whether $\mathsf{pf}$ is a valid opening for the values $\boldsymbol{a} \in \Sigma^{|Q|}$ at the coordinates $Q \subseteq [\ell]$, with respect to the commitment $\mathsf{cm}$.

For a fixed $h$, we say that $(\mathsf{cm}, \mathsf{td})$ *opens* $\boldsymbol{m} \in (\Sigma \cup \{\perp\})^\ell$ if, writing $S := \mathrm{Dom}(\boldsymbol{m})$,

$$\mathsf{VC.Check}^h(\mathsf{cm}, S, \boldsymbol{m}[S], \mathsf{VC.Open}^h(\mathsf{td}, S)) = 1 \ .$$

A VC scheme $\mathsf{VC}[\sigma, \Sigma, \ell]$ satisfies the properties of completeness and extractability, defined as follows.

**Completeness.** For every $h$, message $\boldsymbol{m} \in \Sigma^\ell$, and query set $Q \subseteq [\ell]$, if $(\mathsf{cm}, \mathsf{td}) \leftarrow \mathsf{VC.Commit}^h(\boldsymbol{m})$, then $\mathsf{VC.Check}^h(\mathsf{cm}, Q, \boldsymbol{m}[Q], \mathsf{VC.Open}^h(\mathsf{td}, Q)) = 1$.

**Extractability.** $\mathsf{VC}[\sigma, \Sigma, \ell]$ has *extraction error* $\kappa(\sigma, \ell, q)$ if there exists a deterministic algorithm $\mathsf{VC.Extract}$ that runs in polynomial time, and outputs a string $\boldsymbol{m}' \in (\Sigma \cup \{\perp\})^\ell$ and a trapdoor $\mathsf{td}'$ such that, for every stateful $q$-query ROM adversary $\mathcal{A}$, we have

$$\Pr\left[\begin{array}{l} \mathsf{VC.Check}^h(\mathsf{cm}, Q, \boldsymbol{a}, \mathsf{pf}) = 1 \\ \wedge \begin{pmatrix} \mathsf{VC.Check}^h(\mathsf{cm}, Q, \boldsymbol{a}, \mathsf{pf}') = 0 \\ \vee\ \boldsymbol{m}'[Q] \neq \boldsymbol{a} \end{pmatrix} \end{array} \middle| \begin{array}{l} h \xleftarrow{\$} \mathcal{U}(\sigma) \\ \mathsf{cm} \xleftarrow{\mathsf{qtr}} \mathcal{A}^h(1^\sigma) \\ (\boldsymbol{m}', \mathsf{td}') \leftarrow \mathsf{VC.Extract}(\mathsf{cm}, \mathsf{qtr}) \\ (Q, \boldsymbol{a}, \mathsf{pf}) \leftarrow \mathcal{A}^h(1^\sigma) \\ \mathsf{pf}' \leftarrow \mathsf{VC.Open}^h(\mathsf{td}', Q) \end{array}\right] \leq \kappa(\sigma, \ell, q) \ , \qquad (5)$$

where $\mathsf{qtr}$ contains at most $q$ query-response pairs. If the commitment phase outputs a sequence of commitments $\mathsf{cm}_1, \ldots, \mathsf{cm}_n$, with $\mathsf{qtr}_j$ being the prefix of the trace preceding the output of $\mathsf{cm}_j$, then the analogous bound holds for $(\boldsymbol{m}_j, \mathsf{td}_j) \leftarrow \mathsf{VC.Extract}(\mathsf{cm}_j, \mathsf{qtr}_j)$ with error $\kappa(\sigma, \ell, q, n)$, which we call the *multi-extraction error*. Here the opening phase adaptively chooses $(j, Q, \boldsymbol{a}, \mathsf{pf})$ after all commitments have been output, and the failure event in (5) is evaluated for $(\mathsf{cm}_j, \boldsymbol{m}_j, \mathsf{td}_j)$.

The above definition generalizes the guarantee in [CY24, Lemma 18.5.1]: for Merkle trees, the $\mathsf{VC.Check}^h(\mathsf{cm}, Q, \boldsymbol{a}, \mathsf{pf}') = 0$ condition simplifies to $\mathsf{pf} \neq \mathsf{pf}'$ since Merkle opening proofs are unique (up to finding a hash collision). We also omit a discussion of position binding for VC schemes, since extractability implies position binding as noted in [CY24, Remark 18.5.6].

**The associated VCR scheme.** Given $\mathsf{VC}[\sigma, \Sigma, \ell] := (\mathsf{VC.Commit}, \mathsf{VC.Open}, \mathsf{VC.Check})$, define the corresponding VCR protocols and relations as follows.

- The commitment indexer outputs $(\perp, \perp)$. The prover computes $(\mathsf{cm}, \mathsf{td}) \leftarrow \mathsf{VC.Commit}^h(\boldsymbol{m})$, sends $\mathsf{cm}$ to $\mathsf{Com}.\mathbf{V}$, and retains $(\boldsymbol{m}, \mathsf{td})$ as its witness. The transcript is simply the commitment $\mathsf{cm}$.
- The opening reduction $\mathsf{Rdx}$ is a 1-round IR with a trivial indexer that outputs $(\perp, \perp, \perp)$.[5] On instance $(\mathsf{cm}, Q, \boldsymbol{a})$ and witness $(\boldsymbol{m}, \mathsf{td})$, the prover sends $\mathsf{pf} \leftarrow \mathsf{VC.Open}^h(\mathsf{td}, Q)$, and outputs the witness $\perp$. The verifier outputs the decision $\mathsf{VC.Check}^h(\mathsf{cm}, Q, \boldsymbol{a}, \mathsf{pf})$ and the instance $\perp$.
- The bad relation $\mathcal{R}^h_{\mathsf{bad}}$ is empty, and the output relation $\mathcal{R}_{\mathsf{out}}$ is trivial.
- For a fixed random oracle $h$, the commitment relations are

$$\mathcal{R}^h_{\mathsf{com}} := \left\{(\mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) : (\mathsf{cm}, \mathsf{td}) = \mathsf{VC.Commit}^h(\boldsymbol{m})\right\} \ ,$$
$$\tilde{\mathcal{R}}^h_{\mathsf{com}} := \left\{(\mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) : (\mathsf{cm}, \mathsf{td}) \text{ opens } \boldsymbol{m}\right\} \ .$$

Completeness of VC follows from completeness of Com and Rdx.

**From VCR extractability to VC extractability.** We require the commitment extractor $\mathsf{Com}.\mathbf{E}$ to output *monotone trapdoors*, in the following sense. Consider an arbitrary random oracle $h$ and output $(\boldsymbol{m}', \mathsf{td}')$ of $\mathsf{Com}.\mathbf{E}(\mathsf{cm}, \mathsf{qtr})$. For every $Q \subseteq \mathrm{Dom}(\boldsymbol{m}')$, $(\boldsymbol{m}', \mathsf{td}')$ must satisfy the following:

$$(\mathsf{cm}, \mathsf{td}') \text{ opens } \boldsymbol{m}' \implies \mathsf{VC.Check}^h(\mathsf{cm}, Q, \boldsymbol{m}'[Q], \mathsf{VC.Open}^h(\mathsf{td}', Q)) = 1 \ .$$

[5] Since the indices are trivial, we will omit them from the discussion in this section.

**Lemma 4.10** (VCR extractability implies VC extractability)**.** *Suppose the associated VCR scheme has commitment extractability error $\kappa(\sigma, \ell, q)$ with a polynomial-time extractor* $\mathsf{Com}.\mathbf{E}$ *that outputs monotone trapdoors, and* $\mathsf{Rdx}$ *has round-by-round knowledge soundness error* 0*. Then* $\mathsf{VC}$ *has extraction error at most $\kappa(\sigma, \ell, q + q_{\mathsf{check}})$, where* $\mathsf{VC}.\mathsf{Check}^h$ *makes at most $q_{\mathsf{check}}$ queries.*

*Proof.* The VC extractor $\mathsf{VC}.\mathsf{Extract}(\mathsf{cm}, \mathsf{qtr})$ simply outputs $\mathsf{Com}.\mathbf{E}(\mathsf{cm}, \mathsf{qtr})$. Fix a stateful $q$-query VC adversary $\mathcal{A}_{\mathsf{VC}}$. We construct a VCR commitment adversary $\mathcal{A}_{\mathsf{VCR}}$ whose first stage runs the first stage of $\mathcal{A}_{\mathsf{VC}}$ and outputs the same commitment $\mathsf{cm}$. Consequently, both experiments invoke the same commitment extractor on the same prefix trace $\mathsf{qtr}$, and obtain the same $(\boldsymbol{m}', \mathsf{td}')$.

In its second stage, $\mathcal{A}_{\mathsf{VCR}}$ runs the second stage of $\mathcal{A}_{\mathsf{VC}}$ to obtain $(Q, \boldsymbol{a}, \mathsf{pf})$. It runs $\mathsf{VC}.\mathsf{Check}^h(\mathsf{cm}, Q, \boldsymbol{a}, \mathsf{pf})$ to obtain the opening verifier's trace, and invokes the round-by-round extractor of $\mathsf{Rdx}$ on $\mathsf{pf}$ (the transcript of $\mathsf{Rdx}$) and output witness $\perp$, and thus obtains the extracted witness $(\boldsymbol{m}, \mathsf{td})$, which it then outputs.

Since the RBR knowledge-error of $\mathsf{Rdx}$ is 0, we know that if $\mathsf{VC}.\mathsf{Check}^h(\mathsf{cm}, Q, \boldsymbol{a}, \mathsf{pf}) = 1$, then $((\mathsf{cm}, Q, \boldsymbol{a}), (\boldsymbol{m}, \mathsf{td})) \in \widetilde{\mathsf{Query}}(\tilde{\mathcal{R}}_{\mathsf{com}})^h$, which means that

$$(\mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \tilde{\mathcal{R}}^h_{\mathsf{com}} \text{ and } \boldsymbol{m}[Q] = \boldsymbol{a} \ .$$

Clearly, $\mathcal{A}_{\mathsf{VCR}}$ makes at most $q + q_{\mathsf{check}}$ random-oracle queries. Since $\mathcal{R}^h_{\mathsf{bad}}$ is empty, VCR commitment extractability (Definition 4.3) bounds the probability of

$$(\mathsf{cm}, (\boldsymbol{m}', \mathsf{td}')) \notin \tilde{\mathcal{R}}^h_{\mathsf{com}} \text{ or } \boldsymbol{m} \not\preceq \boldsymbol{m}'$$

by $\kappa(\sigma, \ell, q + q_{\mathsf{check}})$. If this failure event does not occur, then we have $\boldsymbol{m} \preceq \boldsymbol{m}'$, and thus $\mathrm{Dom}(\boldsymbol{m}) \subseteq \mathrm{Dom}(\boldsymbol{m}')$. Thus, given $\boldsymbol{m}[Q] = \boldsymbol{a}$, we also get $\boldsymbol{m}'[Q] = \boldsymbol{a}$ and $Q \subseteq \mathrm{Dom}(\boldsymbol{m}')$. Furthermore, $(\mathsf{cm}, \mathsf{td}')$ opens $\boldsymbol{m}'$ since $(\mathsf{cm}, (\boldsymbol{m}', \mathsf{td}')) \in \tilde{\mathcal{R}}^h_{\mathsf{com}}$. Since $\mathsf{Com}.\mathbf{E}$ outputs monotone trapdoors and $Q \subseteq \mathrm{Dom}(\boldsymbol{m}')$, we finally get $\mathsf{VC}.\mathsf{Check}^h(\mathsf{cm}, Q, \boldsymbol{a}, \mathsf{VC}.\mathsf{Open}^h(\mathsf{td}', Q)) = 1$ as desired. □

## 4.3 Merkle trees

**Definition 4.11.** A Merkle tree $\mathsf{MT} \coloneqq \mathsf{MT}[\sigma, \Sigma, \ell]$ is a VC scheme with the following methods. We assume that $\ell$ is a power of two, padding messages with a fixed symbol otherwise.

- $\mathsf{MT}.\mathsf{Commit}^h(\boldsymbol{m})$: takes as input a message $\boldsymbol{m} \in \Sigma^\ell$. It initializes a binary tree where the $i$-th leaf contains the label $h(\boldsymbol{m}[i])$, and an internal node contains the label $h(L \,\|\, R)$ where $L$ and $R$ are the labels of the left and right children. It outputs a commitment $\mathsf{cm} \in \{0,1\}^\sigma$, which is the root of the tree, and a trapdoor $\mathsf{td} \in \{0,1\}^{O(\sigma\ell)}$ consisting of the entire binary tree.
- $\mathsf{MT}.\mathsf{Open}^h(\mathsf{td}, Q)$: outputs an opening proof $\mathsf{pf} \in \{0,1\}^{|Q|\sigma \log \ell}$ for a query index set $Q \subseteq [\ell]$. $\mathsf{pf}$ is the deduplicated union of the authentication paths for the indices in $Q$, each consisting of the sibling labels on the path from that leaf to the root.
- $\mathsf{MT}.\mathsf{Check}^h(\mathsf{cm}, Q, \boldsymbol{a}, \mathsf{pf})$: outputs a bit $b \in \{0,1\}$ indicating whether $\mathsf{pf}$ is a valid opening for the values $\boldsymbol{a} \in \Sigma^{|Q|}$ at the coordinates $Q \subseteq [\ell]$, with respect to the commitment $\mathsf{cm}$. It uses $\mathsf{pf}$ and the purported values in $Q$ to compute the hash at the root of the Merkle tree, and accepts if and only if it is equal to $\mathsf{cm}$.

**The Merkle extractor.** We use the deterministic extractor of [CY24, Construction 18.5.3], with the following conventions for partial messages. On input $(\mathsf{cm}, \mathsf{qtr})$, it reconstructs the partial binary tree rooted at $\mathsf{cm}$ by following the appropriate pre-images recorded in $\mathsf{qtr}$. The label of a node is recovered only if the label of its parent is already recovered, and if the parents label exists as a response in $\mathsf{qtr}$. The message value of a leaf is recovered only if the label of the node is already recovered and if the label exists as a response in $\mathsf{qtr}$. Any

unrecovered label or message value is written as $\perp$. If the trace contains a collision (distinct queries with the same response), or $\mathsf{cm}$ does not occur as a response, the extractor instead returns a string of $\perp$s for both the message and the trapdoor. We denote this extractor by $\mathsf{MT.Extract}$, which clearly runs in polynomial time in the length of its inputs.

Merkle trees satisfy completeness by [CY24, Lemma 18.2.1]. With the above extractor, they have the following VC extraction and multi-extraction errors by [CY24, Lemmas 18.5.1 and 18.5.7].

$$\kappa_{\mathsf{MT}}(\sigma, \ell, q) \leq \frac{q(q-1)/2 + 2\ell(\log \ell + 1)}{2^\sigma} \ ,$$
$$\kappa_{\mathsf{MT}}(\sigma, \ell, q, n) \leq \frac{3q(q-1)/2 + 2\ell(\log \ell + 1) + (n-1)q}{2^\sigma} \ .$$

We now show that the VCR scheme associated with Merkle trees satisfies the properties required for Lemma 4.10. The first and second statements assert that the Merkle extractor's output has VCR commitment extractability error at most $\kappa_{\mathsf{MT}}(\sigma, \ell, q, n)$ and the Merkle extractor's output trapdoor is monotone. The third statement asserts that the RBR knowledge error of $\mathsf{Rdx}$ is zero.

**Lemma 4.12** (Merkle VCR properties)**.** *Let* $\mathsf{MT} = \mathsf{MT}[\sigma, \Sigma, \ell]$ *be a Merkle tree VC scheme, and let* $(\mathsf{Com}, \mathsf{Rdx}, \mathcal{R}^{\mathcal{U}}_{\mathsf{bad}}, \mathcal{R}^{\mathcal{U}}_{\mathsf{com}}, \mathcal{R}^{\mathcal{U}}_{\mathsf{out}})$ *be the associated VCR scheme.*

*(i) For every* $\mathsf{cm} \in \{0,1\}^\sigma$ *and random-oracle query trace* $\mathsf{qtr}$*, let* $(\boldsymbol{m}, \mathsf{td})$ *be the output of* $\mathsf{MT.Extract}(\mathsf{cm}, \mathsf{qtr})$*. Then* $(\mathsf{cm}, \mathsf{td})$ *opens* $\boldsymbol{m}$*. Furthermore,* $\mathsf{MT.Open}^h(\mathsf{td}, Q)$ *is a valid opening of* $\boldsymbol{m}[Q]$ *for every* $Q \subseteq \mathrm{Dom}(\boldsymbol{m})$*.*

*(ii) Let* $\mathcal{A}$ *be a stateful* $q$*-query ROM adversary whose commitment phase outputs commitments* $\mathsf{cm}_1, \ldots, \mathsf{cm}_n$ *in sequence, with* $\mathsf{qtr}_j$ *the prefix of the trace preceding the output of* $\mathsf{cm}_j$*, and whose opening phase outputs an index* $j \in [n]$*, a partial string* $\boldsymbol{m} \in (\Sigma \cup \{\perp\})^\ell$*, and a trapdoor* $\mathsf{td}$*. Let* $(\boldsymbol{m}_j, \mathsf{td}_j) \leftarrow \mathsf{MT.Extract}(\mathsf{cm}_j, \mathsf{qtr}_j)$ *for every* $j \in [n]$*. The adversary makes at most* $q$ *queries across all stages and chooses* $j$ *adaptively in the opening phase. Then, over* $h \xleftarrow{\$} \mathcal{U}(\sigma)$ *and the randomness of* $\mathcal{A}$*,*

$$\Pr\left[(\mathsf{cm}_j, \mathsf{td}) \text{ opens } \boldsymbol{m} \wedge \boldsymbol{m} \not\preceq \boldsymbol{m}_j\right] \leq \kappa_{\mathsf{MT}}(\sigma, \ell, q, n) \ ,$$

*and the bound* $\kappa_{\mathsf{MT}}(\sigma, \ell, q)$ *applies when* $n = 1$*.*

*(iii) There exists an extractor* $\mathbf{E}_{\mathsf{IR}}$ *that takes as input* $(\mathsf{cm}, Q, \boldsymbol{a})$ *and a Merkle opening proof* $\mathsf{pf}$ *such that* $\mathsf{MT.Check}(\mathsf{cm}, Q, \boldsymbol{a}, \mathsf{pf}) = 1$*, and outputs* $(\boldsymbol{m}, \mathsf{td})$ *such that* $(\mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \tilde{\mathcal{R}}_{\mathsf{com}}$ *and* $\boldsymbol{m}[Q] = \boldsymbol{a}$ *with probability 1.*

*Proof.* The first statement follows directly from the Merkle extractor construction, and the fact that the extractor's output trapdoor is monotone.

For the second statement, apply the VC multi-extraction guarantee above to the adversary that runs $\mathcal{A}$ and then outputs the opening $(j, Q, \boldsymbol{m}[Q], \mathsf{MT.Open}^h(\mathsf{td}, Q))$ with $Q := \mathrm{Dom}(\boldsymbol{m})$. This conversion preserves every commitment-prefix trace and makes no further random-oracle queries since $\mathsf{MT.Open}$ makes none. If $(\mathsf{cm}_j, \mathsf{td})$ opens $\boldsymbol{m}$ then this opening verifies, and $\boldsymbol{m} \not\preceq \boldsymbol{m}_j$ states exactly that $\boldsymbol{m}_j[Q] \neq \boldsymbol{m}[Q]$. Thus, the event in (ii) is contained in the VC extraction-failure event for the adaptively selected commitment. When $n = 1$, use the single-commitment guarantee instead.

The extractor for the third statement outputs $\mathsf{pf}$ as the trapdoor $\mathsf{td}$. For the message it outputs $\boldsymbol{m}$ which is equal to $\boldsymbol{a}$ on $Q$ and $\perp$ everywhere else. Clearly, this extractor succeeds with probability 1. □

Finally, we bound the honest commitment binding error (Definition 4.8) of Merkle trees.

**Lemma 4.13** (Merkle honest commitment binding)**.** *The VCR scheme associated with* $\mathsf{MT}[\sigma, \Sigma, \ell]$ *has honest commitment binding error*

$$\kappa_{\mathsf{hon}}(\sigma, \ell, q) \leq \frac{\binom{q+2\ell+\log \ell}{2}}{2^{\sigma}} \ .$$

*Proof.* Consider the experiment of Definition 4.8 with a $q$-query adversary $\mathcal{A}$. The honest commitment to $\boldsymbol{m}_{\mathsf{hon}}$ makes at most $2\ell - 1$ random-oracle queries. For a well-formed output with $\boldsymbol{m} \not\preceq \boldsymbol{m}_{\mathsf{hon}}$, verify the authentication path at the first disagreeing defined coordinate, using at most $\log \ell + 1$ further queries. A valid witness makes this path verify, since it is the same path checked independently in the opening on $\mathrm{Dom}(\boldsymbol{m})$. Comparing it with the honest tree exposes a collision between distinct random-oracle queries. The combined trace contains at most $q + 2\ell + \log \ell$ queries, so a union bound gives the stated error. □

# 5 Transforming IORs into IRs with VCR schemes

The iBCS transform [BCS16; CY24] turns an IOP into an interactive argument where the prover commits to each proof oracle using Merkle trees. Arc [BMNW25] extends this to a transformation from IORs to IRs. Arc's transformation compiles a state-restoration knowledge-sound (SRKS) IOR from $\mathcal{R}_1$ to $\mathcal{R}_2$ to a SRKS IR from a *committed* form of $\mathcal{R}_1$ to a committed form of $\mathcal{R}_2$. This transformation works only with one specific vector commitment scheme, namely Merkle trees. In this section, we show how to adapt the transformation to instead work with arbitrary VCR schemes, and show that the transformation preserves round-by-round knowledge soundness (RBRKS) instead of SRKS. Note that the transformation does not return a non-interactive reduction of knowledge (NIRK), but we can apply the Fiat–Shamir transformation [FS86] to the output of our transformation to obtain a NIRK that is secure in the ROM (see Appendix C.3).

## 5.1 Committed and bad certificate relations

We discuss *committed relations* with respect to VCR schemes: given an indexed oracle promise relation $\mathcal{R}$, the committed relation of $\mathcal{R}$ is an indexed promise relation which initializes all instance oracles with VCR schemes. This generalizes the notion of committed relations with respect to Merkle trees in Arc [BMNW25].

**Random-oracle notation.** To accommodate different random-oracle output lengths, each VCR scheme $\mathsf{V}_\tau = \mathsf{V}_\tau[\sigma_\tau, \Sigma_\tau, \ell_\tau]$ is assigned a fixed, distinct tag $t_\tau$, and the output length $\sigma_\tau$ is required to satisfy $\sigma_\tau \le \sigma$. Given a master random oracle $h \xleftarrow{\$} \mathcal{U}(\sigma)$, its interface is $h_\tau(s) := h(\langle t_\tau, s\rangle)[\{1, \dots, \sigma_\tau\}]$, where $\langle \cdot, \cdot \rangle$ is an injective encoding. We write $\mathsf{V}_\tau.\mathsf{A}^h$ and $\mathsf{V}_\tau.\mathcal{R}^h$ for the scheme's algorithms and relations evaluated relative to $h_\tau$. This allows us to use a single random oracle $h$ for all VCR schemes, while still allowing each scheme to have its own output length.

For the following definitions, consider an indexed oracle promise relation $\mathcal{R} = (\mathcal{R}, \tilde{\mathcal{R}})$ with $|\vec{\mathbb{y}}|$ instance oracles, and a tuple of VCR schemes

$$\vec{\mathsf{V}} := \left((\mathsf{V}_i.\mathsf{Com}, \mathsf{V}_i.\mathsf{Rdx}, \mathsf{V}_i.\mathcal{R}^{\mathcal{U}}_{\mathsf{bad}}, \mathsf{V}_i.\mathcal{R}^{\mathcal{U}}_{\mathsf{com}}, \mathsf{V}_i.\mathcal{R}^{\mathcal{U}}_{\mathsf{out}})\right)_{i\in[|\vec{\mathbb{y}}|]} \quad .$$

For compactness, write $\vec{\mathbb{i}}_{\mathsf{com}} := (\mathbb{i}_{\mathsf{com},i})_{i\in[|\vec{\mathbb{y}}|]}$, $\vec{\mathsf{cm}} := (\mathsf{cm}_i)_{i\in[|\vec{\mathbb{y}}|]}$, and $\vec{\mathsf{td}} := (\mathsf{td}_i)_{i\in[|\vec{\mathbb{y}}|]}$. Let $\ell_i$ and $\Sigma_i$ be the prescribed length and alphabet of the $i$-th instance oracle, and let $\mathsf{V}_i = \mathsf{V}_i[\sigma_i, \Sigma_i, \ell_i]$ for every $i$.

**Definition 5.1** (committed relations)**.** The *committed relation* $\mathcal{S}^{\mathcal{U}} := \mathsf{CR}[\mathcal{R}, \vec{\mathsf{V}}]^{\mathcal{U}}$ is an indexed promise relation relative to a random oracle $h \xleftarrow{\$} \mathcal{U}(\sigma)$. Membership in either its strict or relaxed relation requires valid commitment indices:

$$\forall i \in [|\vec{\mathbb{y}}|] : \quad \mathbb{i}_{\mathsf{com},i} = \mathsf{V}_i.\mathbb{i}_{\mathsf{com}} \quad , \tag{6}$$

where $\mathsf{V}_i.\mathbb{i}_{\mathsf{com}} = (\mathsf{pk}_i, \mathsf{vk}_i)$ is the pair of keys output by $\mathsf{V}_i.\mathsf{Com}.\mathbf{I}^h(1^{\sigma_i})$, as in Definition 4.1.

- **Strict relation:** A tuple $((\mathbb{i}, \vec{\mathbb{i}}_{\mathsf{com}}), (\mathbb{x}, \vec{\mathsf{cm}}), (\mathbb{w}, \vec{\mathbb{y}}, \vec{\mathsf{td}}))$ belongs to the strict relation $\mathcal{S}^h$ if and only if $\vec{\mathbb{i}}_{\mathsf{com}}$ satisfies (6) and the following conditions hold:

$$\begin{aligned} & (\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}, \mathbb{w}) \in \mathcal{R} \\ \wedge\ & \forall i \in [|\vec{\mathbb{y}}|] : (\mathbb{i}_{\mathsf{com},i}, \mathsf{cm}_i, (\mathbb{y}_i, \mathsf{td}_i)) \in \mathsf{V}_i.\mathcal{R}^h_{\mathsf{com}} \\ \wedge\ & \forall i \in [|\vec{\mathbb{y}}|] : (\mathbb{i}_{\mathsf{com},i}, \mathsf{cm}_i, \mathsf{td}_i) \notin \mathsf{V}_i.\mathcal{R}^h_{\mathsf{bad}} \quad . \end{aligned} \tag{7}$$

- **Relaxed relation:** A tuple $((\mathbb{i}, \vec{\mathbb{i}}_{\mathsf{com}}), (\mathbb{x}, \vec{\mathsf{cm}}), (\mathbb{w}, \vec{\mathbb{y}}, \vec{\mathsf{td}}))$ where $\mathbb{y}_i \in (\Sigma_i \cup \{\bot\})^{\ell_i}$ for every $i$ belongs to the relaxed relation $\tilde{\mathcal{S}}^h$ if and only if $\vec{\mathbb{i}}_{\mathsf{com}}$ satisfies (6) and

$$\begin{aligned} & (\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}, \mathbb{w}) \in \tilde{\mathcal{R}} \\ \wedge\ & \forall i \in [|\vec{\mathbb{y}}|] : (\mathbb{i}_{\mathsf{com},i}, \mathsf{cm}_i, (\mathbb{y}_i, \mathsf{td}_i)) \in \mathsf{V}_i.\tilde{\mathcal{R}}^h_{\mathsf{com}} \\ \wedge\ & \forall i \in [|\vec{\mathbb{y}}|] : (\mathbb{i}_{\mathsf{com},i}, \mathsf{cm}_i, \mathsf{td}_i) \notin \mathsf{V}_i.\mathcal{R}^h_{\mathsf{bad}} \ . \end{aligned} \tag{8}$$

In addition to the committed relation, we also need to define a *bad certificate relation*, which records valid commitment openings whose trapdoors lie in the corresponding bad relation. Its role is explained in detail in Section 5.4, and is necessary to capture VCR schemes with interactive commitment protocols.

**Definition 5.2** (Bad certificate relation)**.** The *bad certificate relation* $\mathcal{S}^{\mathcal{U}} := \mathsf{BCR}[\vec{\mathsf{V}}]^{\mathcal{U}}$ is an indexed promise relation relative to a random oracle $h \xleftarrow{\$} \mathcal{U}(\sigma)$. The strict relation is empty, and a tuple $(\vec{\mathbb{i}}_{\mathsf{com}}, \vec{\mathsf{cm}}, (j, \mathbb{y}_j, \mathsf{td}_j))$ where $j \in [|\vec{\mathbb{y}}|]$ and $\mathbb{y}_j \in (\Sigma_j \cup \{\bot\})^{\ell_j}$ belongs to the relaxed relation $\tilde{\mathcal{S}}^h$ if and only if all commitment indices satisfy (6) and

$$\begin{aligned} & (\mathbb{i}_{\mathsf{com},j}, \mathsf{cm}_j, (\mathbb{y}_j, \mathsf{td}_j)) \in \mathsf{V}_j.\tilde{\mathcal{R}}^h_{\mathsf{com}} \\ \wedge\ & (\mathbb{i}_{\mathsf{com},j}, \mathsf{cm}_j, \mathsf{td}_j) \in \mathsf{V}_j.\mathcal{R}^h_{\mathsf{bad}} \ . \end{aligned} \tag{9}$$

Looking ahead, we will consider the union of the two relations $\mathsf{CR}[\mathcal{R}, \vec{\mathsf{V}}] \cup \mathsf{BCR}[\vec{\mathsf{V}}]$. We first extend $\mathsf{BCR}[\vec{\mathsf{V}}]$ to the common index $(\mathbb{i}, \vec{\mathbb{i}}_{\mathsf{com}})$ and instance $(\mathbb{x}, \vec{\mathsf{cm}})$ by ignoring $\mathbb{i}$ and $\mathbb{x}$. We then take the tagged union: its ordinary witnesses are $(\mathsf{good}, \mathbb{w}, \vec{\mathbb{y}}, \vec{\mathsf{td}})$, and its bad certificates are $(\mathsf{bad}, j, \mathbb{y}_j, \mathsf{td}_j)$.

## 5.2 Parameters and notation

**Oracles.** Consider an IOR $\mathsf{IOR}$ from $\mathcal{R}_{\mathsf{in}}$ to $\mathcal{R}_{\mathsf{out}}$ with $\mathsf{k}$ rounds such that $\mathcal{R}_{\mathsf{in}}$ has $\mathsf{n}_{\mathsf{in}}$ instance oracles and $\mathcal{R}_{\mathsf{out}}$ has $\mathsf{n}_{\mathsf{out}}$ instance oracles. In round $i$, its prover message is $M_i = (\Pi_i, \alpha_i)$, with proof oracle $\Pi_i$ and plaintext message $\alpha_i$ as in Section 3.3. Define the set of all oracles as

$$\Xi_{\mathsf{all}} := \{\mathbb{I}\} \cup \{\mathbb{y}_j : j \in [\mathsf{n}_{\mathsf{in}}]\} \cup \{\Pi_i : i \in [\mathsf{k}]\} \ ,$$

and the set of new oracles as

$$\Xi_{\mathsf{pf}} := \{\Pi_i : i \in [\mathsf{k}]\} \ .$$

We also use $\Xi_{\mathsf{qry}} \subseteq \Xi_{\mathsf{all}}$ to denote a fixed set of oracle labels, known to the indexer, containing every oracle that the IOR verifier may query, and $\Xi_{\mathsf{out}} \subseteq \Xi_{\mathsf{all}} \setminus \{\mathbb{I}\}$ to denote the set of output instance oracles of $\mathcal{R}_{\mathsf{out}}$. We require $\Xi_{\mathsf{all}} = \Xi_{\mathsf{qry}} \cup \Xi_{\mathsf{out}}$: every oracle must be included in the opening reductions, selected as an output oracle, or both.[6] We write $\vec{\mathbb{y}}'_{\mathsf{out}}$ for the ordered tuple of output instance oracles selected by $\mathsf{IOR}$.

**VCR assignment.** For each $\tau \in \Xi_{\mathsf{all}}$, let $\mathsf{V}_\tau = \mathsf{V}_\tau[\sigma_\tau, \Sigma_\tau, \ell_\tau]$ be a VCR scheme for the oracle $\tau$, where $\ell_\tau$ and $\Sigma_\tau$ are the length and alphabet of $\tau$ and the random-oracle output length $\sigma_\tau$ satisfies $\sigma_\tau \leq \sigma$. We assign each entry a fixed, distinct tag $t_\tau$ and use the random-oracle interface convention of Section 5.1. We use the shorthand $\vec{\mathsf{V}}_{\mathsf{all}}$ to denote the tuple of all VCR schemes, and refer to it as the *VCR assignment* for $\mathsf{IOR}$. We also use $\vec{\mathsf{V}}_{\mathsf{in}}$ to denote the tuple of VCR schemes for the input instance oracles, $\vec{\mathsf{V}}_{\mathsf{out}}$ to denote the tuple of VCR schemes for the output instance oracles, $\vec{\mathsf{V}}_{\mathsf{pf}}$ to denote the tuple of VCR schemes for the proof oracles, and $\vec{\mathsf{V}}_{\mathsf{qry}}$ to denote the tuple of VCR schemes for the queried oracles. The tuple $\vec{\mathsf{V}}_{\mathsf{out}}$ follows the order of $\vec{\mathbb{y}}'_{\mathsf{out}}$. The VCR assignment also specifies a fixed verifier challenge from each round's challenge set for the

[6] An oracle in $\Xi_{\mathsf{qry}}$ may have an empty query set in a particular execution, but its opening reduction will still be performed.

commitment to the index oracle $\mathbb{I}$. An empty oracle, with $\ell_\tau = 0$, is assigned the VCR scheme associated with the trivial VC scheme (Section 4.2), whose commitment, trapdoor, and opening proof are empty and whose opening check accepts only empty inputs. This scheme satisfies the assumptions of Theorem 5.3 with all errors zero, and we omit it from error bounds.

**Transformation from IOR to IR.** Given the above notation, the transformation in Fig. 2 takes as input an $\mathsf{IOR}$ from $\mathcal{R}_{\mathsf{in}}$ to $\mathcal{R}_{\mathsf{out}}$ and a tuple of VCR schemes $\vec{\mathsf{V}}_{\mathsf{all}}$ as defined above, and outputs $\mathsf{IR} := \mathsf{T}[\mathsf{IOR}, \vec{\mathsf{V}}_{\mathsf{all}}]$ from the following indexed promise relation:

$$\mathsf{CR}[\mathcal{R}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}} \cup \mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}} \ , \tag{10}$$

to the following indexed promise relation:

$$(\mathsf{CR}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}} \cup \mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}}) \times \vec{\mathsf{V}}_{\mathsf{qry}}.\mathcal{R}^{\mathcal{U}}_{\mathsf{out}} \ . \tag{11}$$

### 5.3 The transformation construction

At a high level, the transformation from IORs to IRs is structured as follows.

1. The IR indexer runs the IOR indexer, the commitment indexer of the VCR scheme for each oracle in $\Xi_{\mathsf{pf}} \cup \{\mathbb{I}\}$, and the reduction indexer of the VCR scheme for each oracle in $\Xi_{\mathsf{qry}}$.

2. In the interaction phase, the IR prover and verifier simulate the IOR prover and verifier respectively. For each IOR prover message $M_i = (\Pi_i, \alpha_i)$, the IR prover forwards $\alpha_i$, and the parties engage in the commitment phase of the VCR scheme for $\Pi_i$.

3. Once the interactive phase of the IOR is finished, for each oracle in $\Xi_{\mathsf{qry}}$, the IR prover simulates the queries of the IOR verifier using the plaintext messages, random challenges, and oracle answers, and sends the resulting query sets and answers to the IR verifier.

4. For each oracle in $\Xi_{\mathsf{qry}}$, the IR prover and verifier run the opening reduction of the VCR scheme with respect to the query sets and answers. At the end of each such interaction, the IR prover and verifier output instances and witnesses for the output relation of the VCR scheme.

**Inputs:** $\mathsf{IOR} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ is an IOR from $\mathcal{R}_{\mathsf{in}}$ to $\mathcal{R}_{\mathsf{out}}$, and $\vec{\mathsf{V}}_{\mathsf{all}}$ is the list of VCR schemes for the oracles of $\mathsf{IOR}$ as defined in Section 5.2.

---

**Part 1: Indexer.** $\mathbf{I}^h_{\mathsf{IR}}(\mathbb{i})$, where $\mathbb{i} = \left(\mathbb{i}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}.\mathbb{i}_{\mathsf{com}}\right)$ is an index for $\mathsf{CR}[\mathcal{R}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}} \cup \mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}}$:

1. Check that all input commitment indices satisfy (6) for their prescribed parameters. On malformed input or a failed check, output $(\perp, \perp, \perp)$ and terminate.
2. Run the IOR indexer: $(\iota, \mathbb{I}, \mathbb{i}_{\mathsf{out}}) \leftarrow \mathbf{I}(\mathbb{i}_{\mathsf{in}})$.
3. Run the VCR commitment indexer for each new oracle. That is, for each $\tau \in \Xi_{\mathsf{pf}} \cup \{\mathbb{I}\}$, compute

$$(\mathsf{V}_\tau.\mathsf{pk}, \mathsf{V}_\tau.\mathsf{vk}) \leftarrow \mathsf{V}_\tau.\mathsf{Com}.\mathbf{I}^h(1^{\sigma_\tau}) \ ,$$

   and let $\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}} := (\mathsf{V}_\tau.\mathsf{pk}, \mathsf{V}_\tau.\mathsf{vk})$.
4. Run the VCR opening indexer for each queried oracle. That is, for each $\tau \in \Xi_{\mathsf{qry}}$, compute

$$(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{out}}, \mathsf{V}_\tau.\mathsf{pk}_{\mathsf{Rdx}}, \mathsf{V}_\tau.\mathsf{vk}_{\mathsf{Rdx}}) \leftarrow \mathsf{V}_\tau.\mathsf{Rdx}.\mathbf{I}^h(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}) \ .$$

5. Simulate $\mathsf{V}_{\mathbb{I}}.\mathsf{Com}^h(\mathsf{V}_{\mathbb{I}}.\mathbb{i}_{\mathsf{com}}, \mathbb{I})$, using the fixed verifier challenges specified by the VCR assignment, to obtain a commitment $\mathsf{V}_{\mathbb{I}}.\mathsf{cm}$ and trapdoor $\mathsf{V}_{\mathbb{I}}.\mathsf{td}$.
6. Output the index for $(\mathsf{CR}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}} \cup \mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}}) \times \vec{\mathsf{V}}_{\mathsf{qry}}.\mathcal{R}_{\mathsf{out}}^{\mathcal{U}}$:

$$(\mathbb{i}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}.\mathbb{i}_{\mathsf{com}}, \vec{\mathsf{V}}_{\mathsf{qry}}.\mathbb{i}_{\mathsf{out}}) \ .$$

7. Output a prover key $\mathsf{pk}_{\mathsf{IR}} := (\mathbb{i}_{\mathsf{in}}, \mathbb{I}, \mathsf{V}_{\mathbb{I}}.\mathsf{td}, \vec{\mathsf{V}}_{\mathsf{pf}}.\mathsf{pk}, \vec{\mathsf{V}}_{\mathsf{qry}}.\mathsf{pk}_{\mathsf{Rdx}})$ and a verifier key $\mathsf{vk}_{\mathsf{IR}} := (\iota, \mathsf{V}_{\mathbb{I}}.\mathsf{cm}, \vec{\mathsf{V}}_{\mathsf{pf}}.\mathsf{vk}, \vec{\mathsf{V}}_{\mathsf{qry}}.\mathsf{vk}_{\mathsf{Rdx}})$ for the IR.

---

**Part 2: Interaction phase.** $\langle \mathbf{P}_{\mathsf{IR}}^h(\mathsf{pk}_{\mathsf{IR}}, \mathbb{x}_{\mathsf{IR}}, \mathbb{w}_{\mathsf{IR}}), \mathbf{V}_{\mathsf{IR}}^h(\mathsf{vk}_{\mathsf{IR}}, \mathbb{x}_{\mathsf{IR}}) \rangle$, where

$$\mathbb{x}_{\mathsf{IR}} = \left(\mathbb{x}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}.\mathsf{cm}\right) \ , \ \mathbb{w}_{\mathsf{IR}} = \left(\mathsf{good}, \mathbb{w}_{\mathsf{in}}, \vec{\mathbb{y}}_{\mathsf{in}}, \vec{\mathsf{td}}_{\mathsf{in}}\right)$$

are an instance and a witness for the strict relation $\mathsf{CR}[\mathcal{R}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}} \cup \mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}}$, and $\mathsf{pk}_{\mathsf{IR}}$ and $\mathsf{vk}_{\mathsf{IR}}$ are the prover and verifier keys output by the indexer $\mathbf{I}_{\mathsf{IR}}^h$. If $\mathsf{vk}_{\mathsf{IR}} = \perp$, the verifier rejects immediately.

1. Simulate the interaction phase of $\mathsf{IOR}$. For $i = 1, \ldots, \mathsf{k}$:
   (a) $\mathbf{P}_{\mathsf{IR}}^h$ runs the $i$-th round of the IOR prover $\mathbf{P}$, which outputs $M_i = (\Pi_i, \alpha_i)$, consisting of a proof oracle and a plaintext message.
   (b) $\mathbf{P}_{\mathsf{IR}}^h$ sends $\alpha_i$ to $\mathbf{V}_{\mathsf{IR}}^h$.
   (c) $\mathbf{P}_{\mathsf{IR}}^h$ and $\mathbf{V}_{\mathsf{IR}}^h$ run $\mathsf{V}_{\Pi_i}.\mathsf{Com}^h(\mathsf{V}_{\Pi_i}.\mathsf{pk}, \mathsf{V}_{\Pi_i}.\mathsf{vk}, \Pi_i)$ to commit to the proof oracle $\Pi_i$. At the end of the commitment interaction, $\mathbf{V}_{\mathsf{IR}}^h$ holds $\mathsf{V}_{\Pi_i}.\mathsf{cm}$ and $\mathbf{P}_{\mathsf{IR}}^h$ holds $(\mathsf{V}_{\Pi_i}.\mathsf{cm}, \mathsf{V}_{\Pi_i}.\mathsf{td})$.
   (d) $\mathbf{V}_{\mathsf{IR}}^h$ samples $\rho_i \xleftarrow{\$} R_i$ and sends it to $\mathbf{P}_{\mathsf{IR}}^h$, which forwards it to $\mathbf{P}$.
2. Simulate the query phase of $\mathsf{IOR}$.
   (a) $\mathbf{P}_{\mathsf{IR}}^h$ runs $\mathbf{I}(\mathbb{i}_{\mathsf{in}})$ to recover $(\iota, \mathbb{I}, \mathbb{i}_{\mathsf{out}})$, and simulates

$$\mathbf{V}^{\mathbb{I}, \vec{\mathbb{y}}_{\mathsf{in}}, \vec{\Pi}}(\iota, \mathbb{x}_{\mathsf{in}}; \vec{\alpha}, \vec{\rho}) \ ,$$

   and obtains the set $Q_\tau \subseteq [\ell_\tau]$ of positions queried for each oracle $\tau \in \Xi_{\mathsf{qry}}$.
   (b) $\mathbf{P}_{\mathsf{IR}}^h$ sets $\boldsymbol{a}_\tau := \tau[Q_\tau]$ for each $\tau \in \Xi_{\mathsf{qry}}$ and sends the tuple $(Q_\tau, \boldsymbol{a}_\tau)_{\tau \in \Xi_{\mathsf{qry}}}$ to $\mathbf{V}_{\mathsf{IR}}^h$.
   (c) $\mathbf{V}_{\mathsf{IR}}^h$ runs $\mathbf{V}$ on explicit inputs $(\iota, \mathbb{x}_{\mathsf{in}}; \vec{\alpha}, \vec{\rho})$, answering each query to position $j$ of oracle $\tau$ with $\boldsymbol{a}_\tau[j]$. It rejects if any supplied query set or answer has an invalid domain or alphabet, or if the execution requests a position outside $Q_\tau$ (or an oracle outside $\Xi_{\mathsf{qry}}$). Otherwise, it obtains the IOR decision $b_{\mathsf{IOR}} \in \{0, 1\}$ and short output instance $\mathbb{x}'_{\mathsf{out}}$.
3. For each oracle $\tau \in \Xi_{\mathsf{qry}}$, $\mathbf{P}_{\mathsf{IR}}^h$ and $\mathbf{V}_{\mathsf{IR}}^h$ run the following IR:

$$(b_\tau, \mathsf{V}_\tau.\mathbb{x}_{\mathsf{out}}, \mathsf{V}_\tau.\mathbb{w}_{\mathsf{out}}) \leftarrow \mathsf{V}_\tau.\mathsf{Rdx}^h((\mathsf{V}_\tau.\mathsf{pk}_{\mathsf{Rdx}}, \mathsf{V}_\tau.\mathsf{vk}_{\mathsf{Rdx}}), (\mathsf{V}_\tau.\mathsf{cm}, Q_\tau, \boldsymbol{a}_\tau), (\tau, \mathsf{V}_\tau.\mathsf{td})) \ ,$$

   where $\mathsf{V}_\tau.\mathbb{x}_{\mathsf{out}}$ and $b_\tau \in \{0, 1\}$ are output to $\mathbf{V}_{\mathsf{IR}}^h$, and $\mathsf{V}_\tau.\mathbb{w}_{\mathsf{out}}$ is output to $\mathbf{P}_{\mathsf{IR}}^h$.
4. $\mathbf{P}_{\mathsf{IR}}^h$ completes the simulated IOR prover's computation to obtain the output witness $\mathbb{w}'_{\mathsf{out}}$.
5. $\mathbf{V}_{\mathsf{IR}}^h$ computes the overall decision of the IR as

$$b_{\mathsf{IR}} := b_{\mathsf{IOR}} \wedge \bigwedge_{\tau \in \Xi_{\mathsf{qry}}} b_\tau \ ,$$

   and rejects if $b_{\mathsf{IR}} = 0$.

6. If $b_{\mathsf{IR}} = 1$, $\mathbf{P}^h_{\mathsf{IR}}$ outputs the following witness:

$$\mathbb{w}'_{\mathsf{IR}} := \left((\mathsf{good}, \mathbb{w}'_{\mathsf{out}}, \vec{\mathbb{y}}'_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}.\mathsf{td}), \vec{\mathsf{V}}_{\mathsf{qry}}.\mathbb{w}_{\mathsf{out}}\right) \ ,$$

and $\mathbf{V}^h_{\mathsf{IR}}$ outputs the following instance:

$$\mathbb{x}'_{\mathsf{IR}} := \left(\mathbb{x}'_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}.\mathsf{cm}, \vec{\mathsf{V}}_{\mathsf{qry}}.\mathbb{x}_{\mathsf{out}}\right) \ .$$

**Figure 2:** $\mathsf{T}[\mathsf{IOR}, \vec{\mathsf{V}}_{\mathsf{all}}]$, an interactive reduction between the relations defined in Eqs. (10) and (11). The specialization to non-bad extracted input openings is stated in Section 5.4.

**Theorem 5.3** (Transformation from IORs to IRs)**.** *Let* $\mathsf{IOR}$ *be a complete* $\mathsf{k}$*-round IOR from* $\mathcal{R}_{\mathsf{in}}$ *to* $\mathcal{R}_{\mathsf{out}}$*, with round-by-round knowledge errors* $(\kappa_{\mathsf{IOR},j})_{j\in[\mathsf{k}]}$ *and extraction times* $(\mathsf{et}_{\mathsf{IOR},j})_{j\in[\mathsf{k}]}$*. Assume that the ordered output-oracle selection contains no repeated labels: each* $\tau \in \Xi_{\mathsf{out}}$ *is selected exactly once. Assume that* $\mathcal{R}_{\mathsf{out}}$ *and the IOR knowledge states are strongly monotone (Definitions 3.3 and 3.13). Consider the VCR assignment described in Section 5.2, and for each* $\tau \in \Xi_{\mathsf{all}}$*, assume that:*

1. $\mathsf{V}_\tau.\mathsf{Com}$ *is complete (Definition 4.2) and extractable (Definition 4.3) with error* $\kappa_{\mathsf{com},\tau}$ *and extraction time* $\mathsf{et}_{\mathsf{com},\tau}$*.*
2. $\mathsf{V}_\tau.\mathcal{R}^{\mathcal{U}}_{\mathsf{bad}}$ *has round-by-round sparsity errors* $(\kappa_{\mathsf{sp},\tau,i})_{i\in[\mathsf{V}_\tau.\mathsf{k}_{\mathsf{com}}]}$ *(see Definition 4.4).*
3. $\mathsf{V}_\tau.\mathsf{Rdx}$ *is complete and has round-by-round knowledge errors* $(\kappa_{\mathsf{rdx},\tau,i})_{i\in[\mathsf{V}_\tau.\mathsf{k}_{\mathsf{rdx}}]}$ *and extraction times* $(\mathsf{et}_{\mathsf{rdx},\tau,i})_{i\in[\mathsf{V}_\tau.\mathsf{k}_{\mathsf{rdx}}]}$ *(see Definition 3.17).*
4. *There is a deterministic polynomial-time algorithm* $\mathsf{DecBad}_\tau(\mathbb{i}, \mathsf{cm}, \mathsf{td}, \mathsf{qtr})$ *such that for every ROM-adversary that outputs* $(\mathbb{i}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \mathsf{V}_\tau.\tilde{\mathcal{R}}^h_{\mathsf{com}}$ *with random-oracle trace* $\mathsf{qtr}$*, then*

$$\mathsf{DecBad}_\tau(\mathbb{i}, \mathsf{cm}, \mathsf{td}, \mathsf{qtr}) = 1 \iff (\mathbb{i}, \mathsf{cm}, \mathsf{td}) \in \mathsf{V}_\tau.\mathcal{R}^h_{\mathsf{bad}} \ .$$

5. *The index-oracle VCR scheme* $\mathsf{V}_{\mathbb{I}}$ *has honest commitment binding error at most* $\kappa_{\mathsf{hon},\mathbb{I}}$*.*

*Then* $\mathsf{IR} := \mathsf{T}[\mathsf{IOR}, \vec{\mathsf{V}}_{\mathsf{all}}]$ *in Fig. 2 is a complete IR from* $\mathsf{CR}[\mathcal{R}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}} \cup \mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}}$ *to* $(\mathsf{CR}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}} \cup \mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}}) \times \vec{\mathsf{V}}_{\mathsf{qry}}.\mathcal{R}^{\mathcal{U}}_{\mathsf{out}}$*. It has*

$$\mathsf{k}_{\mathsf{IR}} := \mathsf{k} + \sum_{\tau\in\Xi_{\mathsf{pf}}} \mathsf{V}_\tau.\mathsf{k}_{\mathsf{com}} + \sum_{\tau\in\Xi_{\mathsf{qry}}} \mathsf{V}_\tau.\mathsf{k}_{\mathsf{rdx}}$$

*rounds. To state its error and time bounds, for each* $j \in [\mathsf{k}]$*, define the set of oracles available to the IOR verifier in round* $j$*:*

$$S_0 := \{\mathbb{y}_i : i \in [\mathsf{n}_{\mathsf{in}}]\}, \qquad S_j := S_0 \cup \{\Pi_i : i \in [j]\} \ .$$

*For compiled statement size at most* $n$ *and a* $q$*-query ROM adversary, let* $n_*$*,* $t_*$*, and* $q_* \geq q$ *be polynomial bounds in* $(\sigma, n, q)$ *on component statement sizes, auxiliary running time of the compiled extractor excluding component extractor calls, and total random-oracle queries in each component security reduction, respectively. The query bound* $q_*$ *includes the adversary's queries and queries made in simulated indexer and verifier computations. Suppress parameters below: evaluate IOR bounds at* $n_*$*, commitment extraction, honest commitment binding, and sparsity bounds at* $(\sigma_\tau, \ell_\tau, q_*)$*, and opening-reduction bounds at* $(\sigma_\tau, n_*, q_*)$*. The*

*round-by-round knowledge soundness errors can be bounded by*

$$\kappa_{\mathsf{IR},r} \leq \begin{cases} \kappa_{\mathsf{sp},\tau,s}, & r \text{ is round } s \text{ of } \mathsf{V}_\tau.\mathsf{Com},\ \tau \in \Xi_{\mathsf{pf}}, \\ \kappa_{\mathsf{IOR},j} + \sum_{\tau \in S_j} \kappa_{\mathsf{com},\tau}, & r \text{ is IOR round } j < \mathsf{k}, \\ \kappa_{\mathsf{IOR},\mathsf{k}} + 2\sum_{\tau \in S_\mathsf{k}} \kappa_{\mathsf{com},\tau} + \kappa_{\mathsf{hon},\mathbb{I}}, & r \text{ is IOR round } \mathsf{k}, \\ \kappa_{\mathsf{rdx},\tau,s}, & r \text{ is round } s \text{ of } \mathsf{V}_\tau.\mathsf{Rdx},\ \tau \in \Xi_{\mathsf{qry}}. \end{cases}$$

*The corresponding extraction times satisfy*

$$\mathsf{et}_{\mathsf{IR},r} \leq t_* + \begin{cases} 0, & r \text{ is a commitment round}, \\ \mathsf{et}_{\mathsf{IOR},j} + \sum_{\tau \in S_j} \mathsf{et}_{\mathsf{com},\tau}, & r \text{ is IOR round } j \in [\mathsf{k}], \\ \mathsf{et}_{\mathsf{rdx},\tau,s}, & r \text{ is round } s \text{ of } \mathsf{V}_\tau.\mathsf{Rdx}. \end{cases}$$

***Efficiency.*** *Let* $\mathsf{it}_{\mathsf{IOR}}$, $\mathsf{pt}_{\mathsf{IOR}}$ *and* $\mathsf{vt}_{\mathsf{IOR}}$ *be the IOR indexer, prover and verifier times, and let* $\mathsf{len}_{\mathsf{IOR}}$ *bound its total plaintext prover communication* $\sum_{j=1}^{\mathsf{k}} |\alpha_j|$ *in bits. For each oracle* $\tau$*, let* $\mathsf{it}_{\mathsf{com},\tau}$, $\mathsf{pt}_{\mathsf{com},\tau}$, $\mathsf{vt}_{\mathsf{com},\tau}$, *and* $\mathsf{len}_{\mathsf{com},\tau}$ *be the indexer time, prover time, verifier time, and proof size of* $\mathsf{V}_\tau.\mathsf{Com}$*, and use subscripts* $\mathsf{rdx},\tau$ *for the corresponding quantities of* $\mathsf{V}_\tau.\mathsf{Rdx}$*. Let* $\mathsf{len}_{\mathsf{qry}}$ *bound the bit length of the query record* $(Q_\tau, \boldsymbol{a}_\tau)_{\tau \in \Xi_{\mathsf{qry}}}$*. Assuming constant-time random access to all stored inputs, the IR indexer time, interaction-phase prover and verifier times, and proof size satisfy*

$$\mathsf{it}_{\mathsf{IR}} \leq O\left(\mathsf{it}_{\mathsf{IOR}} + \sum_{\tau \in \Xi_{\mathsf{all}}} \mathsf{it}_{\mathsf{com},\tau} + \sum_{\tau \in \Xi_{\mathsf{qry}}} \mathsf{it}_{\mathsf{rdx},\tau} + \mathsf{pt}_{\mathsf{com},\mathbb{I}} + \mathsf{vt}_{\mathsf{com},\mathbb{I}}\right),$$

$$\mathsf{pt}_{\mathsf{IR}} \leq O\left(\mathsf{pt}_{\mathsf{IOR}} + \mathsf{it}_{\mathsf{IOR}} + \mathsf{vt}_{\mathsf{IOR}} + \mathsf{len}_{\mathsf{qry}} + \sum_{\tau \in \Xi_{\mathsf{pf}}} \mathsf{pt}_{\mathsf{com},\tau} + \sum_{\tau \in \Xi_{\mathsf{qry}}} \mathsf{pt}_{\mathsf{rdx},\tau}\right),$$

$$\mathsf{vt}_{\mathsf{IR}} \leq O\left(\mathsf{vt}_{\mathsf{IOR}} + \mathsf{len}_{\mathsf{qry}} + \sum_{\tau \in \Xi_{\mathsf{pf}}} \mathsf{vt}_{\mathsf{com},\tau} + \sum_{\tau \in \Xi_{\mathsf{qry}}} \mathsf{vt}_{\mathsf{rdx},\tau}\right),$$

$$\mathsf{len}_{\mathsf{IR}} \leq \mathsf{len}_{\mathsf{IOR}} + \mathsf{len}_{\mathsf{qry}} + \sum_{\tau \in \Xi_{\mathsf{pf}}} \mathsf{len}_{\mathsf{com},\tau} + \sum_{\tau \in \Xi_{\mathsf{qry}}} \mathsf{len}_{\mathsf{rdx},\tau}.$$

Completeness, indexer time, prover time, verifier time, and proof size directly follow from the construction in Fig. 2. The knowledge soundness proof appears in Appendix A.1.

### 5.4 Eliminating the bad certificate relation

While the output of the transformation is an IR from $\mathsf{CR}[\mathcal{R}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}} \cup \mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}}$ to $(\mathsf{CR}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}} \cup \mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}}) \times \vec{\mathsf{V}}_{\mathsf{qry}}.\mathcal{R}_{\mathsf{out}}^{\mathcal{U}}$ in Theorem 5.3, we also discuss a specialization of it which returns an IR from $\mathsf{CR}[\mathcal{R}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}} \times \vec{\mathsf{V}}_{\mathsf{qry}}.\mathcal{R}_{\mathsf{out}}^{\mathcal{U}}$, thus omitting the bad certificate relations from the input and the output.

**The purpose of the bad certificate relation.** Consider an input instance oracle $\tau \in \vec{\mathbb{y}}_{\mathsf{in}}$. The IR output by the transformation in Fig. 2 assumes that $\tau$ has already been committed to, so the input instance of the

IR must already contain a commitment $\mathsf{cm}_\tau$, allowing the knowledge soundness adversary to output an arbitrary $\mathsf{cm}_\tau$. In particular, it can output a commitment $\mathsf{cm}_\tau$ with a valid bad opening: there exist $\boldsymbol{m}_\tau, \mathsf{td}_\tau$ with $(\mathbb{i}_\tau, \mathsf{cm}_\tau, (\boldsymbol{m}_\tau, \mathsf{td}_\tau)) \in \mathsf{V}_\tau.\tilde{\mathcal{R}}^h_{\mathsf{com}}$ and $(\mathbb{i}_\tau, \mathsf{cm}_\tau, \mathsf{td}_\tau) \in \mathsf{V}_\tau.\mathcal{R}^h_{\mathsf{bad}}$. In this case, VCR extractability (Definition 4.3) guarantees a valid extracted commitment witness, but permits its trapdoor to be bad, in which case $\boldsymbol{m}_\tau \preceq \boldsymbol{m}'_\tau$ does not need to hold where $\boldsymbol{m}'_\tau$ is the extracted message. Since the IR extractor needs this extension guarantee to recover a witness for the underlying relation, it cannot always guarantee an extracted witness in $\mathsf{CR}[\mathcal{R}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}]$. Thus, we have to allow the IR extractor to output a bad certificate for the corresponding input instance oracle, which is captured by the relaxed relation of $\mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}}$. Note that when every $\mathsf{V}_i.\mathcal{R}_{\mathsf{bad}}$ is empty (like in the case of VC schemes), there is no need to define $\mathsf{BCR}[\mathcal{R}, \vec{\mathsf{V}}]^{\mathcal{U}}$, and is thus only required to capture VCR schemes with interactive commitment protocols.

**Motivation for elimination of bad certificate relation.** While the above argument motivates the need for the bad certificate relation, the IR output by the transformation in Fig. 2 does not accurately capture the non-oracle analogue of the input IOR. In particular, before the interactive phase of the IOR begins, its verifier already has access to fixed input oracles, and the probability that the oracle's answers for a string $\tau$ are incorrect is at most $\beta_\tau^{\mathsf{IOR}} = 0$. To model this in the compiled IR, we can consider an upper bound $\beta_\tau$ on the probability of the following event: the extraction adversary outputs a commitment $\mathsf{cm}_\tau$ to $\tau$ such that the valid opening extracted from $\mathsf{cm}_\tau$ (and the corresponding random-oracle query trace) by the VCR extractor has a bad trapdoor. If the above event does not occur, then VCR extractability as defined in Definition 4.3 guarantees the above required extension guarantee, which gives us an IR from $\mathsf{CR}[\mathcal{R}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}} \times \vec{\mathsf{V}}_{\mathsf{qry}}.\mathcal{R}^{\mathcal{U}}_{\mathsf{out}}$, with the RBR knowledge soundness errors adjusted by the $\beta_\tau$s.

**Small values of $\boldsymbol{\beta_\tau}$.** In general, the value of $\beta_\tau$ can be as large as 1 if $\mathsf{V}_\tau.\mathcal{R}_{\mathsf{bad}}$ is non-empty. However, looking ahead to the sections where we construct SNARKs (Section 7) and accumulation schemes (Section 8) from IORs and VCR schemes by applying the transformation in Fig. 2, we can get smaller values of $\beta_\tau$ in many circumstances:

- Any IOR considered while constructing a SNARK is a component of an IOP, which does not have input oracles by definition. Thus, the input oracles of an IOR in this case are the proof oracles sent during the interactive phase of another IOR. If all these IORs are transformed into IRs with Fig. 2, then the commitment to each oracle string was generated by an interaction between the resulting IR prover and verifier. In this case, we have
$$\beta_\tau = \sum_{s=1}^{\mathsf{V}_\tau.\mathsf{k}_{\mathsf{com}}} \kappa_{\mathsf{sp},\tau,s} \ ,$$
which follows from the definition of round-by-round sparsity (see Definition 4.4).
- An IOR considered while constructing an accumulation scheme does not necessarily have to be a component of an IOP, and requires handling of arbitrary input oracles. So it may seem like the compiled IR may have adversarially chosen commitments with $\beta_\tau$ as large as 1. However, one can 'force' an arbitrarily chosen commitment to look like one that was produced via interaction with a verifier, by enforcing that the challenges in the VCR commitment must be consistent with the Fiat–Shamir transform (i.e., must be obtained by hashing the transcript of the commitment interaction so far). By doing so, we can once again show that $\beta_\tau = \sum_{s=1}^{\mathsf{V}_\tau.\mathsf{k}_{\mathsf{com}}} \kappa_{\mathsf{sp},\tau,s}$ as desired. We discuss this in more detail in Remark 8.6.

Given the above, we state the following specialization of the generic transformation theorem.

**Theorem 5.4** (Specialized transformation)**.** *Consider the notation established and assumptions made in Theorem 5.3. In addition, for each input oracle string $\tau \in S_0$, let $\beta_\tau$ be an upper bound on the probability of*

*the following event: the output of* $\mathsf{V}_\tau.\mathsf{Com}.\mathbf{E}$ *on the input commitment and the full prior query trace belongs in* $\mathsf{V}_\tau.\tilde{\mathcal{R}}^h_{\mathsf{com}}$*, and the trapdoor belongs in* $\mathsf{V}_\tau.\mathcal{R}^h_{\mathsf{bad}}$*. If the bound holds for arbitrary commitment prefixes of the compiled IR with arbitrary completions, with all prior queries included in the trace and all queries contributing to the query bound* $q_*$*, then the output of Fig. 2 is a complete IR from*

$$\mathsf{CR}[\mathcal{R}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}} \quad \textit{to} \quad \mathsf{CR}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}} \times \vec{\mathsf{V}}_{\mathsf{qry}}.\mathcal{R}^{\mathcal{U}}_{\mathsf{out}} \ ,$$

*and the RBRKS error* $\kappa_{\mathsf{com},\tau}$ *is replaced with* $\kappa_{\mathsf{com},\tau} + \beta_\tau$ *for each input oracle* $\tau \in S_0$*. The round complexity, extraction times, indexer, prover and verifier times, and proof-size bounds are those of Theorem 5.3.*

The proof appears in Appendix A.2.

# 6 Canopy: a VCR scheme from high-distance codes

We describe Canopy, the VCR scheme that underlies our improvements to hash-based SNARKs (Section 7.2) and hash-based accumulation schemes (Section 8.3).

As illustrated in Fig. 1, Canopy first splits the message into $k$ blocks and commits to each block with a shallow Merkle tree. It then maps the block roots into the field through $\mathsf{H2F}$, obtaining the *canopy message* $\boldsymbol{y} \in \mathbb{F}^k$, and commits to its encoding under $\mathcal{C}$ with a top Merkle tree rooted at $\mathsf{rt}$. For a dishonest prover, the word committed under $\mathsf{rt}$ may lie near the encodings of several canopy messages, since the proximity parameter $\delta$ may exceed $\delta_{\mathcal{C}}/2$. To distinguish these messages, the verifier samples a point $\boldsymbol{z}_{\mathsf{ood}}$ and the prover answers with the evaluation $\hat{y}(\boldsymbol{z}_{\mathsf{ood}})$ of the message's multilinear extension. This is the out-of-domain sampling approach of [ACFY24]: except with small probability, at most one nearby message matches the claimed evaluation. For valid commitment witnesses, the bad relation records the exceptional case in which distinct nearby messages match.

To open the coordinates in $Q$, the prover sends the root of each touched block together with the authentication paths inside that block. Thus, each queried coordinate costs at most $d+1$ hash values, compared with $m$ for a Merkle opening, and the prover does not open any paths in the top tree. It remains to check that $\boldsymbol{y}[v] = \mathsf{H2F}(\mathsf{rt}_v)$ for every touched block $v$. A random challenge $\gamma$ combines these equalities with the out-of-domain evaluation into a single sumcheck claim about the word committed under $\mathsf{rt}$. The resulting claim belongs to $\mathsf{SEP}[\mathcal{C}, \delta]$, whose batching protocol lets us combine the openings of many commitments into one proximity claim (Appendix D).

**Parameters.** Canopy is parameterized by a random-oracle output length $\sigma$, an alphabet $\Sigma$, a message length $\ell = 2^m$, a linear code $\mathcal{C}$, and a proximity parameter $\delta$. The code has parameters $(\mathbb{F}, b, k, \delta_{\mathcal{C}})$, where $k \le \ell$ and both $k$ and $b$ are powers of two. We require $\delta \in (0, \delta_{\mathcal{C}}]$ and $\delta < 1$. The field satisfies $|\mathbb{F}| \ge 2^\sigma$ and comes with a fixed injection $\mathsf{H2F} : \{0,1\}^\sigma \to \mathbb{F}$ whose image can be decided and whose inverse can be computed in polynomial time. We write $\mathsf{Canopy}[\sigma, \Sigma, \ell, \mathcal{C}, \delta]$ for the resulting scheme. A message whose length is not a power of two is padded with a fixed symbol to the next power of two, as for Merkle trees. The *canopy depth* is $d := m - \log k$, so that the message splits into $k$ blocks of $2^d$ leaves. Canopy uses two Merkle commitment schemes from Definition 4.11, both with the same random oracle $h \xleftarrow{\$} \mathcal{U}(\sigma)$: $\mathsf{MT}_{\mathsf{btm}} := \mathsf{MT}[\sigma, \Sigma, 2^d]$ for the blocks and $\mathsf{MT}_{\mathsf{top}} := \mathsf{MT}[\sigma, \mathbb{F}, b]$ for the codeword. A commitment-trapdoor pair opens a partial string *under* one of these schemes when it does so in the sense of Section 4.2 for that scheme.

**Indexing.** We index the coordinates of a message by leaves $u \in \{0,1\}^m$ and write every leaf as $u = v \,\|\, s$, where $v \in \{0,1\}^{m-d}$ is its block index and $s \in \{0,1\}^d$ is its offset. For $r \in \{m, d, m-d\}$, we identify a string $u \in \{0,1\}^r$ with the coordinate $j \in [2^r]$ for which $\langle j-1 \rangle = u$, so that vectors of length $2^r$ are indexed by $\{0,1\}^r$. For $\boldsymbol{m} \in (\Sigma \cup \{\perp\})^\ell$, the block $\boldsymbol{m}_v := (\boldsymbol{m}[v \,\|\, s])_{s \in \{0,1\}^d}$ is the restriction of $\boldsymbol{m}$ to the leaves with block index $v$.

**Construction 6.1** (Canopy)**.** Canopy is the VCR scheme

$$\mathsf{Canopy}[\sigma, \Sigma, \ell, \mathcal{C}, \delta] := \left(\mathsf{Com}, \mathsf{Rdx}, \mathcal{R}^{\mathcal{U}}_{\mathsf{bad}}, \mathcal{R}^{\mathcal{U}}_{\mathsf{com}}, \mathcal{R}^{\mathcal{U}}_{\mathsf{out}}\right)$$

whose components are defined below. Throughout, the random oracle is $h \xleftarrow{\$} \mathcal{U}(\sigma)$, the keys output by $\mathsf{Com}.\mathbf{I}$ are $\mathsf{pk}_{\mathsf{com}} = \mathsf{vk}_{\mathsf{com}} = (\mathcal{C}, \delta)$, and $\mathbb{i}_{\mathsf{com}} = (\mathsf{pk}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{com}})$. Every commitment in the relations below must parse as $(\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}}) \in \{0,1\}^\sigma \times \mathbb{F}^{m-d} \times \mathbb{F}$. We parse every trapdoor as

$$\mathsf{td} = \left(\{(\mathsf{rt}_v, \mathsf{td}_v)\}_{v \in \{0,1\}^{m-d}}, \boldsymbol{y}, \boldsymbol{c}, \mathsf{td}_{\mathsf{top}}\right) \quad ,$$

where $\mathsf{rt}_v \in \{0,1\}^\sigma$, $\boldsymbol{y} \in \mathbb{F}^{2^{m-d}}$, and $\boldsymbol{c} \in (\mathbb{F} \cup \{\perp\})^b$, and no tuple whose trapdoor does not parse in this form belongs to any of the relations below. We write $\hat{y}$ for the multilinear extension of $\boldsymbol{y}$, so that $\hat{y}(v) = \boldsymbol{y}[v]$ for $v \in \{0,1\}^{m-d}$, and likewise $\hat{y}'$, $\hat{y}''$, and $\hat{y}^*$ for $\boldsymbol{y}'$, $\boldsymbol{y}''$, and $\boldsymbol{y}^*$.

**Commitment protocol.** The indexer $\mathsf{Com}.\mathbf{I}$ and the interaction between $\mathsf{Com}.\mathbf{P}$ and $\mathsf{Com}.\mathbf{V}$ are given in Fig. 3. The interaction has two rounds, with $\alpha_1 := \mathsf{rt}$, $\rho_1 := \boldsymbol{z}_{\mathsf{ood}} \in \mathbb{F}^{m-d}$, $\alpha_2 := \zeta_{\mathsf{ood}}$, and an empty second challenge $\rho_2$. We write $\mathsf{cm} = (\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}})$, suppressing the final empty challenge.

**Canopy commitment indexer** $\mathsf{Com}.\mathbf{I}^h(1^\sigma)$:

1. Output $\mathsf{pk}_{\mathsf{com}} := \mathsf{vk}_{\mathsf{com}} := (\mathcal{C}, \delta)$.

**Canopy commitment protocol** $\langle \mathsf{Com}.\mathbf{P}^h(\mathsf{pk}_{\mathsf{com}}, \boldsymbol{m}), \mathsf{Com}.\mathbf{V}^h(\mathsf{vk}_{\mathsf{com}}) \rangle$

1. $\mathsf{Com}.\mathbf{P}$: for every $v \in \{0,1\}^{m-d}$, compute $(\mathsf{rt}_v, \mathsf{td}_v) \leftarrow \mathsf{MT}_{\mathsf{btm}}.\mathsf{Commit}^h(\boldsymbol{m}_v)$.
2. $\mathsf{Com}.\mathbf{P}$: define $\boldsymbol{y} \in \mathbb{F}^{2^{m-d}}$ by $\boldsymbol{y}[v] := \mathsf{H2F}(\mathsf{rt}_v)$ for every $v \in \{0,1\}^{m-d}$.
3. $\mathsf{Com}.\mathbf{P}$: compute $\boldsymbol{c} := \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y})$ and $(\mathsf{rt}, \mathsf{td}_{\mathsf{top}}) \leftarrow \mathsf{MT}_{\mathsf{top}}.\mathsf{Commit}^h(\boldsymbol{c})$.
4. $\mathsf{Com}.\mathbf{P} \to \mathsf{Com}.\mathbf{V}$: $\mathsf{rt}$.
5. $\mathsf{Com}.\mathbf{V} \to \mathsf{Com}.\mathbf{P}$: $\boldsymbol{z}_{\mathsf{ood}} \xleftarrow{\$} \mathbb{F}^{m-d}$.
6. $\mathsf{Com}.\mathbf{P} \to \mathsf{Com}.\mathbf{V}$: $\zeta_{\mathsf{ood}} := \hat{y}(\boldsymbol{z}_{\mathsf{ood}})$.
7. $\mathsf{Com}.\mathbf{P}$ outputs the message $\boldsymbol{m}$ and the trapdoor $\mathsf{td} := \left(\{(\mathsf{rt}_v, \mathsf{td}_v)\}_{v \in \{0,1\}^{m-d}}, \boldsymbol{y}, \boldsymbol{c}, \mathsf{td}_{\mathsf{top}}\right)$.

**Figure 3:** The Canopy commitment indexer and commitment protocol. The commitment is the transcript $\mathsf{cm} = (\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}})$.

**Commitment relation.** The strict commitment relation consists of the commitments and trapdoors that are computed honestly from the committed message:

$$\mathcal{R}^h_{\mathsf{com}} := \left\{ (\mathbb{i}_{\mathsf{com}}, (\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}}), (\boldsymbol{m}, \mathsf{td})) : \begin{array}{rl} & \boldsymbol{m} \in \Sigma^\ell \\ \wedge & \forall v \in \{0,1\}^{m-d} : \\ & \quad (\mathsf{rt}_v, \mathsf{td}_v) = \mathsf{MT}_{\mathsf{btm}}.\mathsf{Commit}^h(\boldsymbol{m}_v) \\ \wedge & \forall v \in \{0,1\}^{m-d} : \boldsymbol{y}[v] = \mathsf{H2F}(\mathsf{rt}_v) \\ \wedge & \boldsymbol{c} = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y}) \\ \wedge & (\mathsf{rt}, \mathsf{td}_{\mathsf{top}}) = \mathsf{MT}_{\mathsf{top}}.\mathsf{Commit}^h(\boldsymbol{c}) \\ \wedge & \zeta_{\mathsf{ood}} = \hat{y}(\boldsymbol{z}_{\mathsf{ood}}) \end{array} \right\}$$

The relaxed commitment relation replaces recomputation by opening and exact encoding by $\delta$-proximity, and

uses partial messages and codewords.

$$\tilde{\mathcal{R}}^{h}_{\mathsf{com}} \coloneqq \left\{ (\mathbb{i}_{\mathsf{com}}, (\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}}), (\boldsymbol{m}, \mathsf{td})) : \begin{array}{r} \boldsymbol{m} \in (\Sigma \cup \{\perp\})^{\ell} \\ \wedge\ \forall v \in \{0,1\}^{m-d} \text{ s.t. } \mathrm{Dom}(\boldsymbol{m}_v) \neq \varnothing : \\ \boldsymbol{y}[v] = \mathsf{H2F}(\mathsf{rt}_v) \\ \wedge\ (\mathsf{rt}_v, \mathsf{td}_v) \text{ opens } \boldsymbol{m}_v \text{ under } \mathsf{MT}_{\mathsf{btm}} \\ \wedge\ \boldsymbol{c} \sim_{\delta} \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y}) \\ \wedge\ (\mathsf{rt}, \mathsf{td}_{\mathsf{top}}) \text{ opens } \boldsymbol{c} \text{ under } \mathsf{MT}_{\mathsf{top}} \\ \wedge\ \zeta_{\mathsf{ood}} = \hat{y}(\boldsymbol{z}_{\mathsf{ood}}) \end{array} \right\}$$

The strict relation is contained in the relaxed one, since an honestly computed Merkle commitment opens its message for every $h$ (Section 4.3) and $\delta > 0$.

**Bad relation.** The bad relation consists of the commitment-trapdoor pairs whose out-of-domain point fails to separate the canopy message from another message whose encoding lies near $\boldsymbol{c}$:

$$\mathcal{R}^{h}_{\mathsf{bad}} \coloneqq \left\{ (\mathbb{i}_{\mathsf{com}}, (\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}}), \mathsf{td}) : \exists \boldsymbol{y}' \in \mathbb{F}^{2^{m-d}} : \begin{array}{r} \boldsymbol{y}' \neq \boldsymbol{y} \\ \wedge\ \boldsymbol{c} \sim_{\delta} \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y}') \\ \wedge\ \hat{y}'(\boldsymbol{z}_{\mathsf{ood}}) = \hat{y}(\boldsymbol{z}_{\mathsf{ood}}) \end{array} \right\}$$

**Output relation.** The output relation is the Merkle-committed sumcheck-evaluation proximity relation

$$\mathcal{R}^{\mathcal{U}}_{\mathsf{out}} \coloneqq \mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}_{\mathsf{top}}]^{\mathcal{U}} \ ,$$

the committed relation of Definition 5.1 in which the single instance oracle of $\mathsf{SEP}[\mathcal{C}, \delta]$, the word $\boldsymbol{w} \in \mathbb{F}^{b}$, is committed with $\mathsf{MT}_{\mathsf{top}}$ viewed as a VCR scheme with index $\mathbb{i}_{\mathsf{MT}} = (\perp, \perp)$ (Section 4.2). Here $\mathsf{MT}_{\mathsf{top}}$ retains the random-oracle interface used for the top commitment, with no additional domain-separation tag. Since $\mathsf{MT}_{\mathsf{top}}$ has an empty bad relation, a relaxed output witness has the form $(\boldsymbol{y}, \boldsymbol{c}, \mathsf{td}_{\mathsf{top}})$, where $\boldsymbol{c}$ may be partial. For an instance $((f, \zeta), \mathsf{rt})$, it is valid exactly when $(\mathsf{rt}, \mathsf{td}_{\mathsf{top}})$ opens $\boldsymbol{c}$ under $\mathsf{MT}_{\mathsf{top}}$ and $((\mathcal{C}, \delta), (f, \zeta), \boldsymbol{c}, \boldsymbol{y})$ lies in the relaxed relation of $\mathsf{SEP}[\mathcal{C}, \delta]$.

**Opening reduction.** For a query set $Q \subseteq [\ell]$, define the set of touched block indices and their offset sets

$$V(Q) \coloneqq \left\{ v \in \{0,1\}^{m-d} : \exists s \in \{0,1\}^{d},\ v \,\|\, s \in Q \right\} \ , \ S_v \coloneqq \left\{ s \in \{0,1\}^{d} : v \,\|\, s \in Q \right\} \ ,$$

and let $v_1 < \cdots < v_t$ be the elements of $V(Q)$ in lexicographic order, where $t \coloneqq |V(Q)|$. For $S \subseteq \{0,1\}^{d}$, we write $\boldsymbol{a}[v \,\|\, S] \coloneqq (\boldsymbol{a}[v \,\|\, s])_{s \in S}$. The indexer $\mathsf{Rdx}.\mathbf{I}$ and the interaction between $\mathsf{Rdx}.\mathbf{P}$ and $\mathsf{Rdx}.\mathbf{V}$ are given in Fig. 4. The interaction has one round, whose prover message has length at most $|Q| \cdot \sigma \cdot (d+1)$ bits and whose challenge is a uniformly random element $\gamma$ of $\mathbb{F}$.

$\mathsf{Rdx}.\mathbf{I}^{h}(\mathbb{i})$:

1. Output $\mathbb{i}_{\mathsf{out}} \coloneqq \mathsf{pk} \coloneqq \mathsf{vk} \coloneqq \perp$ unless $\mathbb{i} = \mathbb{i}_{\mathsf{com}}$.
2. Otherwise, output $\mathbb{i}_{\mathsf{out}} \coloneqq ((\mathcal{C}, \delta), \mathbb{i}_{\mathsf{MT}})$ and $\mathsf{pk} \coloneqq \mathsf{vk} \coloneqq \mathbb{i}_{\mathsf{com}}$.

$\langle \mathsf{Rdx}.\mathbf{P}^{h}(\mathsf{pk}, (\mathsf{cm}, Q, \boldsymbol{a}), (\boldsymbol{m}, \mathsf{td})), \mathsf{Rdx}.\mathbf{V}^{h}(\mathsf{vk}, (\mathsf{cm}, Q, \boldsymbol{a})) \rangle$:

1. Both parties parse $\mathsf{cm} = (\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}})$ and compute $v_1 < \cdots < v_t$ and $(S_{v_i})_{i \in [t]}$ from $Q$. The

verifier rejects malformed inputs or messages, requiring $\mathsf{pf}_i \in \{0,1\}^{|S_{v_i}|\sigma d}$.

2. $\mathsf{Rdx}.\mathbf{P} \to \mathsf{Rdx}.\mathbf{V}$: $\{(\mathsf{rt}_{v_i}, \mathsf{pf}_i)\}_{i\in[t]}$, where $\mathsf{pf}_i \leftarrow \mathsf{MT}_{\mathsf{btm}}.\mathsf{Open}^h(\mathsf{td}_{v_i}, S_{v_i})$.
3. $\mathsf{Rdx}.\mathbf{V} \to \mathsf{Rdx}.\mathbf{P}$: $\gamma \xleftarrow{\$} \mathbb{F}$.
4. Both parties compute $f_\gamma(\boldsymbol{X}, Z) \coloneqq Z \cdot \Big(\mathrm{eq}(\boldsymbol{X}, \boldsymbol{z}_{\mathsf{ood}}) + \sum_{i\in[t]} \gamma^i \cdot \mathrm{eq}(\boldsymbol{X}, v_i)\Big)$ and $\zeta_\gamma \coloneqq \zeta_{\mathsf{ood}} + \sum_{i\in[t]} \gamma^i \cdot \mathsf{H2F}(\mathsf{rt}_{v_i})$, where $f_\gamma$ is represented by $(\boldsymbol{z}_{\mathsf{ood}}, \gamma, (v_i)_{i\in[t]})$.
5. $\mathsf{Rdx}.\mathbf{V}$ outputs $(b, \mathbb{x}_{\mathsf{out}})$, where $\mathbb{x}_{\mathsf{out}} \coloneqq ((f_\gamma, \zeta_\gamma), \mathsf{rt})$. It sets $b \coloneqq 1$ exactly when $\mathsf{vk} \neq \bot$ and every block opening verifies:

$$\forall i \in [t] : \mathsf{MT}_{\mathsf{btm}}.\mathsf{Check}^h(\mathsf{rt}_{v_i}, S_{v_i}, \boldsymbol{a}[v_i \,\|\, S_{v_i}], \mathsf{pf}_i) = 1 \ .$$

6. $\mathsf{Rdx}.\mathbf{P}$ outputs $\mathbb{w}_{\mathsf{out}} \coloneqq (\boldsymbol{y}, \boldsymbol{c}, \mathsf{td}_{\mathsf{top}})$.

**Figure 4:** The Canopy opening indexer and opening reduction. The output instance is $\mathbb{x}_{\mathsf{out}} = ((f_\gamma, \zeta_\gamma), \mathsf{rt})$.

**Efficiency measures.** We measure costs in random-oracle queries and field operations, counting each evaluation of $\mathsf{H2F}$ as one field operation. We write $T_{\mathcal{C}}$ for the number of field operations performed by $\mathsf{Enc}_{\mathcal{C}}$. The protocols in Figs. 3 and 4 have the following costs.

- *Prover cost of* $\mathsf{Com}$*:* Constructing the block trees and the top tree requires $O(\ell + b)$ random-oracle queries. Computing the canopy message, its encoding, and the evaluation $\hat{y}(\boldsymbol{z}_{\mathsf{ood}})$ requires $O(k+T_{\mathcal{C}})$ field operations. Compared with a Merkle commitment, which makes $O(\ell)$ queries, Canopy adds the cost of encoding and constructing the top tree.
- *Verifier cost of* $\mathsf{Com}$*:* The verifier samples $O(\log k)$ field elements; it makes no random-oracle queries and performs no field operations.
- *Proof size of* $\mathsf{Com}$*:* The prover sends one hash value and one field element. Including the verifier's challenge, the commitment $(\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}})$ has $O(\sigma + \log(k) \cdot \log|\mathbb{F}|)$ bits.
- *Prover cost of* $\mathsf{Rdx}$*:* The prover reads $O(|Q| \cdot (d+1))$ hash values from its trapdoor without making random-oracle queries. Computing $\zeta_\gamma$ requires $O(|Q|)$ field operations.
- *Verifier cost of* $\mathsf{Rdx}$*:* The verifier checks the $|Q|$ authentication paths using $O(|Q| \cdot (d+1))$ random-oracle queries, compared with $O(|Q| \cdot m)$ for a Merkle opening. Computing $\zeta_\gamma$ requires $O(|Q|)$ field operations.
- *Proof size of* $\mathsf{Rdx}$*:* The proof consists of at most $|Q|$ block roots and $|Q|$ authentication paths, each containing $d$ hash values, for a total of $O(|Q| \cdot (d+1) \cdot \sigma)$ bits.

**Completeness and soundness.** Let $\mathcal{C}$ be a code that admits a deterministic list-decoding algorithm that, on input $\boldsymbol{w} \in \mathbb{F}^b$, outputs $\Lambda(\mathcal{C}, \boldsymbol{w}, \delta)$ in time $T_{\mathsf{LD}}$ without random-oracle queries. For a partial input, we complete undefined coordinates with a fixed field element, decode the completion, and retain only codewords at distance less than $\delta$ from the original input. In particular, the maximum list size $|\Lambda(\mathcal{C}, \delta)|$ of Definition 3.6 is at most $T_{\mathsf{LD}}$. We use the Merkle extraction errors of Section 4.3 and the extractor guarantees of Lemma 4.12.

**Lemma 6.2.** *The commitment protocol* $\mathsf{Com}$ *of* $\mathsf{Canopy}[\sigma, \Sigma, \ell, \mathcal{C}, \delta]$ *is complete in the sense of Definition 4.2.*

**Lemma 6.3.** *The commitment protocol* $\mathsf{Com}$ *of* $\mathsf{Canopy}[\sigma, \Sigma, \ell, \mathcal{C}, \delta]$ *has extractability error* $\kappa(\sigma, \ell, q) \leq \kappa_{\mathsf{MT}}(\sigma, b, q) + \kappa_{\mathsf{MT}}(\sigma, 2^d, q, 2^{m-d})$ *with extraction time* $T_{\mathsf{LD}} \cdot \mathrm{poly}(\sigma, \ell, b, \log|\mathbb{F}|, \log|\Sigma|, q)$.

**Lemma 6.4.** *The bad relation* $\mathcal{R}_{\mathsf{bad}}^{\mathcal{U}}$ *of* $\mathsf{Canopy}[\sigma, \Sigma, \ell, \mathcal{C}, \delta]$ *has round-by-round sparsity errors*

$$\kappa_{\mathsf{sp},1}(\sigma, \ell, q) \leq \kappa_{\mathsf{MT}}(\sigma, b, q) + \binom{|\Lambda(\mathcal{C}, \delta)|}{2} \cdot \frac{m-d}{|\mathbb{F}|} \qquad \textit{and} \qquad \kappa_{\mathsf{sp},2}(\sigma, \ell, q) = 0 \ .$$

**Lemma 6.5.** *The opening reduction* $\mathsf{Rdx}$ *of* $\mathsf{Canopy}[\sigma, \Sigma, \ell, \mathcal{C}, \delta]$ *is a complete interactive reduction from* $\mathsf{Query}[\mathcal{R}_{\mathsf{com}}]^{\mathcal{U}}$ *to* $\mathcal{R}_{\mathsf{out}}^{\mathcal{U}}$ *in the sense of Definition 3.15.*

**Lemma 6.6.** *The opening reduction* $\mathsf{Rdx}$ *of* $\mathsf{Canopy}[\sigma, \Sigma, \ell, \mathcal{C}, \delta]$ *has round-by-round knowledge soundness error* $\kappa_1(\sigma, n, q) \leq \kappa_{\mathsf{MT}}(\sigma, b, q) + |\Lambda(\mathcal{C}, \delta)| \cdot \frac{n}{|\mathbb{F}|} + \kappa_{\mathsf{MT}}(\sigma, 2^d, q, n)$ *in its single round, in the sense of Definition 3.17, and extraction time* $\mathrm{poly}(\sigma, \ell, b, \log|\mathbb{F}|, \log|\Sigma|, q)$.

**Lemma 6.7.** *There is a deterministic algorithm* $\mathsf{DecBad}$ *that, on input* $(\mathtt{i}, \mathsf{cm}, \mathsf{td}, \mathsf{qtr})$ *and ignoring* $\mathsf{qtr}$*, decides whether* $(\mathtt{i}, \mathsf{cm}, \mathsf{td}) \in \mathcal{R}_{\mathsf{bad}}^{h}$ *in time* $T_{\mathsf{LD}} \cdot \mathrm{poly}(\sigma, \ell, b, \log|\mathbb{F}|)$ *without querying the random oracle.*

**Lemma 6.8.** *The scheme* $\mathsf{Canopy}[\sigma, \Sigma, \ell, \mathcal{C}, \delta]$ *has honest commitment binding error*

$$\kappa_{\mathsf{hon}}(\sigma, \ell, q) \leq \frac{\binom{q+\ell(d+3)+b(\log b+3)}{2}}{2^{\sigma}} \ ,$$

*in the sense of Definition 4.8.*

Proofs of the above seven lemmas appear in Appendix B. Together, they allow us to use the Canopy VCR scheme to instantiate proof and index oracles in the IOR-to-IR transformation of Theorem 5.3.

# 7 Transforming IORs into SNARKs with VCR schemes

The transformation of Section 5 turns an IOR into an interactive reduction whose output consists of committed oracle claims and VCR opening claims. To obtain a non-interactive argument of knowledge, we first prove these remaining claims with an interactive argument, then apply the Fiat–Shamir transformation to the composed protocol. In Section 7.1, we describe this construction for an arbitrary VCR assignment and prove its knowledge soundness in the ROM. In Section 7.2, we instantiate it with Canopy for the IORs underlying modern IOPs for NP. The resulting claims share a code and can be proved together, giving the construction used in Section 9.

## 7.1 The generic transformation

**Setup.** We use the notation of Section 5.2. Throughout this section, $\mathsf{T}$ denotes the specialization of Theorem 5.4, with good omitted from the input witness and the committed component of the output witness. Consider a $\mathsf{k}$-round IOR $\mathsf{IOR}$ from an indexed relation $\mathcal{R}$ to an indexed oracle promise relation $\mathcal{R}_{\mathsf{out}}$. Since $\mathcal{R}$ has no instance oracles, we have $\Xi_{\mathsf{all}} = \{\mathbb{I}\} \cup \Xi_{\mathsf{pf}}$. The set $\Xi_{\mathsf{out}} \subseteq \Xi_{\mathsf{pf}}$ consists of the oracles that $\mathsf{IOR}$ selects as output, each selected exactly once as in Theorem 5.3, and $\Xi_{\mathsf{qry}}$ contains every oracle that the IOR verifier may query, so that $\Xi_{\mathsf{all}} = \Xi_{\mathsf{qry}} \cup \Xi_{\mathsf{out}}$. Let $\vec{\mathsf{V}}_{\mathsf{all}}$ be a VCR assignment for $\mathsf{IOR}$, so that $\vec{\mathsf{V}}_{\mathsf{in}}$ is empty, and let

$$\mathcal{R}^{\mathcal{U}}_{\mathsf{tgt}} \coloneqq \mathsf{CR}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}} \times \vec{\mathsf{V}}_{\mathsf{qry}}.\mathcal{R}^{\mathcal{U}}_{\mathsf{out}}$$

be the output relation of $\mathsf{T}[\mathsf{IOR}, \vec{\mathsf{V}}_{\mathsf{all}}]$. We view an interactive or non-interactive argument as a reduction to the trivial relation $\mathcal{R}_{\mathsf{true}} = \{(\bot, \mathsf{accept}, \bot)\}$ (Remark 3.14 and Appendix C.3). Since $\mathcal{R}$ has no instance oracles, we identify $\mathsf{CR}[\mathcal{R}, \vec{\mathsf{V}}_{\mathsf{in}}]^{\mathcal{U}}$ with $\mathcal{R}$ and suppress trivial factors throughout. Thus $\mathsf{T}[\mathsf{IOR}, \vec{\mathsf{V}}_{\mathsf{all}}]$ is an IR from $\mathcal{R}$ to $\mathcal{R}^{\mathcal{U}}_{\mathsf{tgt}}$.

**Construction 7.1.** Let $\mathsf{IA}$ be an interactive argument for $\mathcal{R}^{\mathcal{U}}_{\mathsf{tgt}}$. The non-interactive argument $\mathsf{ARG}$ for $\mathcal{R}$ is obtained as follows.

1. Apply this specialization to obtain the IR $\mathsf{IR}_1 \coloneqq \mathsf{T}[\mathsf{IOR}, \vec{\mathsf{V}}_{\mathsf{all}}]$ from $\mathcal{R}$ to $\mathcal{R}^{\mathcal{U}}_{\mathsf{tgt}}$.
2. Compose $\mathsf{IR}_1$ with $\mathsf{IA}$ (Appendix C.2) to obtain the interactive argument $\mathsf{IR} \coloneqq \mathsf{IR}_1 \circ \mathsf{IA}$ for $\mathcal{R}$.
3. Apply the hash-chain Fiat–Shamir transformation (Appendix C.3) to obtain $\mathsf{ARG} \coloneqq \mathsf{HCFS}[\mathsf{IR}]$.

The prover of $\mathsf{ARG}$ simulates the IOR prover: for each $M_i = (\Pi_i, \alpha_i)$, it forwards the plaintext message $\alpha_i$ and commits to the proof oracle $\Pi_i$ with the assigned commitment protocol. It then sends the IOR verifier's query sets and answers, runs the assigned opening reductions, and proves the resulting claims with $\mathsf{IA}$. The challenges of the interactive argument are derived from the hash chain. The index commitment uses the fixed challenges specified by the VCR assignment.

**Theorem 7.2** (Transformation from IORs to non-interactive arguments)**.** *Let $\mathsf{IOR}$ be a complete $\mathsf{k}$-round IOR from $\mathcal{R}$ to $\mathcal{R}_{\mathsf{out}}$ with round-by-round knowledge errors $(\kappa_{\mathsf{IOR},j})_{j\in[\mathsf{k}]}$ and extraction times $(\mathsf{et}_{\mathsf{IOR},j})_{j\in[\mathsf{k}]}$, and assume that $\mathcal{R}_{\mathsf{out}}$ and the knowledge states of $\mathsf{IOR}$ are strongly monotone. Consider a VCR assignment $\vec{\mathsf{V}}_{\mathsf{all}}$ for $\mathsf{IOR}$ such that every $\mathsf{V}_\tau$ satisfies assumptions 1–4 of Theorem 5.3 and $\mathsf{V}_{\mathbb{I}}$ satisfies the honest commitment binding assumption (item 5). Let $\mathsf{IA}$ be a complete $\mathsf{k}_{\mathsf{IA}}$-round interactive argument for $\mathcal{R}^{\mathcal{U}}_{\mathsf{tgt}}$ with round-by-round knowledge errors $(\kappa_{\mathsf{IA},s})_{s\in[\mathsf{k}_{\mathsf{IA}}]}$ and extraction times $(\mathsf{et}_{\mathsf{IA},s})_{s\in[\mathsf{k}_{\mathsf{IA}}]}$. Then $\mathsf{ARG}$ in Construction 7.1 is a complete non-interactive argument for $\mathcal{R}$.*

*Let $\mathsf{k}_1$ be the number of rounds of $\mathsf{IR}_1 \coloneqq \mathsf{T}[\mathsf{IOR}, \vec{\mathsf{V}}_{\mathsf{all}}]$, and let $\mathsf{k}_{\mathsf{IR}} \coloneqq \mathsf{k}_1 + \mathsf{k}_{\mathsf{IA}}$. For each round $r \in [\mathsf{k}_{\mathsf{IR}}]$, write $\kappa_r$ and $\mathsf{et}_r$ for the corresponding component bounds of $\mathsf{IR}_1$ or $\mathsf{IA}$, with the statement-size and query-budget*

*substitutions prescribed by Lemmas C.4 and C.5. Then* ARG *has straight-line knowledge error*

$$\kappa_{\mathsf{ARG}}(\sigma, n, q) \leq (q+1) \cdot \sum_{r \in [\mathsf{k_{IR}}]} \kappa_r + \frac{(q + \mathsf{k_{IR}} + 1)^2}{2^\sigma}$$

*and extraction time* $\sum_{r \in [\mathsf{k_{IR}}]} \mathsf{et}_r$ *plus* $O(\mathsf{k_{IR}})$ *executions of the indexer and verifier of* $\mathsf{IR}_1 \circ \mathsf{IA}$*, under the trace-processing convention of Remark 3.18.*

*Proof.* Each step of Construction 7.1 preserves completeness by Theorem 5.4 and Appendices C.2 and C.3. We prove knowledge soundness in the same order.

First, Theorem 5.4 applies by the strong monotonicity and VCR assumptions and the output-oracle selection in the setup; its input-extraction hypothesis is vacuous here. It gives an RBRKS IR $\mathsf{IR}_1$ from $\mathcal{R}$ to $\mathcal{R}^{\mathcal{U}}_{\mathsf{tgt}}$, with $S_j = \{\Pi_i : i \in [j]\}$ because there are no input instance oracles.

Second, Lemmas C.4 and C.5 show that $\mathsf{IR} := \mathsf{IR}_1 \circ \mathsf{IA}$ is an interactive argument for $\mathcal{R}$ with straight-line state-restoration knowledge error at most $(q+1)\sum_r \kappa_r$. The component bounds account for queries made in the simulated indexer and verifier computations, and the bounds of IA use the size of the statement output by $\mathsf{IR}_1$, as specified in those lemmas.

Finally, after padding challenges as in Appendix C.3, Lemma C.7 gives the claimed Fiat–Shamir error. The extraction time follows by adding the component costs and the replay overheads of the same lemmas. Since an accepting output has the trivial witness $\perp$, the resulting reduction extractor is a knowledge extractor for $\mathcal{R}$. □

**Succinctness.** Fiat–Shamir preserves the interactive argument's total prover communication and adds one hash per round to verification. It therefore yields a SNARK when these costs are succinct and the knowledge error is negligible.

### 7.2 An optimized transformation using Canopy

All known constructions of concretely efficient IOPs for an NP-complete relation $\mathcal{R}$ are obtained by composing an IOR from $\mathcal{R}$ to $\mathsf{SEP}[\mathcal{C}, \delta]$, for some code $\mathcal{C}$ and proximity parameter $\delta$, with an IOPP $\mathsf{SEP}[\mathcal{C}, \delta]$ (we discuss this blueprint in more detail in Appendix D.4). In this section, we present an optimized way to use Canopy to compile such IOPs into SNARKs.

At a high level, we apply Canopy to the queried oracles of the first IOR, where the lower opening cost is most useful, and set the code used in Canopy and the code used in $\mathsf{SEP}[\mathcal{C}, \delta]$ to be the same. This choice makes every opening claim a $\mathsf{SEP}[\mathcal{C}, \delta]$ claim about a Merkle-committed word, matching the output claim of the IOR. We prove all these claims together by compiling the batching IOPP BatchPrx of Appendix D with Merkle trees.

**Setup.** Let $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, b, k, \delta_{\mathcal{C}})$ and let $\delta \in (0, \delta_{\mathcal{C}})$. Consider an IOR IOR from an indexed relation $\mathcal{R}$ to $\mathsf{SEP}[\mathcal{C}, \delta]$ with an empty index oracle, as in Appendix D.4. Label its nonempty proof oracles $\Pi_1, \ldots, \Pi_{\eta+1}$, where $\Pi_i$ has alphabet $\Sigma_i$ and padded length $n_i$ for $i \in [\eta]$. The remaining oracle $\Pi_{\eta+1} \in \mathbb{F}^b$ is the output oracle and is not queried. Thus $\Xi_{\mathsf{out}} = \{\Pi_{\eta+1}\}$, and $\Xi_{\mathsf{qry}}$ consists of $\Pi_1, \ldots, \Pi_\eta$ together with the index oracle and the empty proof oracles of the remaining rounds. Empty oracles receive the trivial scheme of Section 5.2, which we omit below, including for BatchPrx.

**VCR assignment.** For $i \in [\eta]$, let $\mathsf{V}_{\Pi_i} := \mathsf{Canopy}[\sigma, \Sigma_i, n_i, \mathcal{C}, \delta]$ (Construction 6.1), which requires $k \leq n_i$, and let $\mathsf{V}_{\Pi_{\eta+1}} := \mathsf{MT}_{\mathcal{C}} := \mathsf{MT}[\sigma, \mathbb{F}, b]$. For $i \in [\eta]$, Construction 6.1 identifies the output relation of $\mathsf{V}_{\Pi_i}$

with $\mathsf{CR}[\mathsf{SEP}[\mathcal{C},\delta],\mathsf{MT}_{\mathcal{C}}]^{\mathcal{U}}$. The committed output claim $\mathsf{CR}[\mathsf{SEP}[\mathcal{C},\delta],\vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}}$ has the same form. Hence the output relation of $\mathsf{T}[\mathsf{IOR},\vec{\mathsf{V}}_{\mathsf{all}}]$ is

$$\mathcal{R}^{\mathcal{U}}_{\mathsf{tgt}} = \left(\mathsf{CR}[\mathsf{SEP}[\mathcal{C},\delta],\mathsf{MT}_{\mathcal{C}}]^{\mathcal{U}}\right)^{\eta+1} \quad ,$$

which is the committed relation $\mathsf{CR}[\mathsf{SEP}[\mathcal{C},\delta]^{\eta+1},\mathsf{MT}_{\mathcal{C}}^{\eta+1}]^{\mathcal{U}}$ of the batch relation of Appendix D up to a reordering of components. Here the $i$-th copy of $\mathsf{MT}_{\mathcal{C}}$ is evaluated relative to the random oracle of $\mathsf{V}_{\Pi_i}$, and we tag the Merkle trees assigned to the instance oracles of BatchPrx below so that they use these same oracles. The schemes assigned to the index and proof oracles of BatchPrx use fresh, distinct tags.

**Construction 7.3.** Let Prx be an IOPP for $\mathsf{MEP}[\mathcal{C},\delta]$ and let BatchPrx be the IOPP for $\mathsf{SEP}[\mathcal{C},\delta]^{\eta+1}$ of Fig. 9. Let $\vec{\mathsf{V}}^{\mathsf{MT}}_{\mathsf{all}}$ be the VCR assignment for BatchPrx that assigns $\mathsf{MT}_{\mathcal{C}}$ to its $\eta+1$ instance oracles, tagged as above, and a Merkle tree of the appropriate parameters to its index oracle and to every proof oracle, and let

$$\mathsf{IA} \coloneqq \mathsf{T}[\mathsf{BatchPrx},\vec{\mathsf{V}}^{\mathsf{MT}}_{\mathsf{all}}] \quad ,$$

which is an interactive argument for $\mathsf{CR}[\mathsf{SEP}[\mathcal{C},\delta]^{\eta+1},\mathsf{MT}_{\mathcal{C}}^{\eta+1}]^{\mathcal{U}} = \mathcal{R}^{\mathcal{U}}_{\mathsf{tgt}}$. The non-interactive argument ARG for $\mathcal{R}$ is Construction 7.1 applied to IOR, $\vec{\mathsf{V}}_{\mathsf{all}}$, and IA.

**Theorem 7.4.** *Assume the following:*

- $|\mathbb{F}| \geq 2^{\sigma}$ *with an injection* H2F *as in Section 6, $b$ is a power of two, and $k \leq n_i$ for every $i \in [\eta]$.*
- *$\mathcal{C}$ admits a deterministic polynomial-time list decoder at radius $\delta$, and $\delta$ and $|\mathbb{F}|$ satisfy the conditions of Lemma D.8 for batch size $\eta+1$.*
- IOR *is complete and round-by-round knowledge sound with strongly monotone knowledge states.*
- Prx *is a complete IOPP for* $\mathsf{MEP}[\mathcal{C},\delta]$ *that satisfies the conditions of Lemma D.8 and has strongly monotone knowledge states.*

*Then* ARG *in Construction 7.3 is a complete non-interactive argument for $\mathcal{R}$ with the straight-line knowledge error and extraction time of Theorem 7.2. Here the round-by-round errors of* $\mathsf{T}[\mathsf{IOR},\vec{\mathsf{V}}_{\mathsf{all}}]$ *are those of Theorem 5.4 with the bounds of Lemmas 6.3, 6.4 and 6.6 for the Canopy schemes and of Section 4.3 for the Merkle trees, and the round-by-round errors of* IA *are those of Theorem 5.4 applied to* BatchPrx *with the bounds of Lemma D.8 and of Section 4.3.*

*Proof.* The Canopy schemes satisfy assumptions 1–3 of Theorem 5.3 by Lemmas 6.2 to 6.6, and assumption 4 by Lemma 6.7 and the polynomial-time list-decoding assumption. The Merkle trees satisfy assumptions 1 and 3 by Section 4.3, and assumptions 2 and 4 because their bad relation is empty. The honest commitment binding assumption (item 5) holds with error zero because the index oracle of IOR is empty. The remaining conditions for the IOR-to-IR transformation hold because $\mathsf{SEP}[\mathcal{C},\delta]$ is strongly monotone and IOR selects a single output oracle.

By Lemma D.8, BatchPrx is complete and RBRKS, with strongly monotone knowledge states inherited from Prx. Its index oracle is inherited from Prx; when nonempty, its Merkle commitment satisfies honest commitment binding by Lemma 4.13. Since the input Merkle schemes have empty bad relations, Theorem 5.4 applies with $\beta_\tau = 0$ for every input oracle and gives a complete RBRKS argument IA for the committed batch relation $\mathcal{R}^{\mathcal{U}}_{\mathsf{tgt}}$. The claim follows from Theorem 7.2. □

# 8 Constructing hash-based accumulation schemes using Canopy

In this section, we show how to construct hash-based accumulation schemes using Canopy. The section is organized as follows:

- In Section 8.1, we recall the formal definition of hash-based accumulation schemes, and review how prior work has constructed hash-based accumulation schemes from IORs.
- In Section 8.2, we discuss a prerequisite ingredient that we will require: an interactive accumulation scheme for $\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}}$.
- In Section 8.3, we present our new Canopy-based approach for constructing hash-based accumulation schemes.

## 8.1 Accumulation schemes in the random oracle model

A tuple of algorithms $\mathsf{ACC} = (\mathbf{I}_{\mathsf{ACC}}, \mathbf{P}_{\mathsf{ACC}}, \mathbf{V}_{\mathsf{ACC}}, \mathbf{D}_{\mathsf{ACC}})$ is an accumulation scheme in the random oracle model for a (ternary) relation $\mathcal{R}$ if it satisfies completeness and knowledge soundness as defined below. An accumulator is a pair $\mathsf{acc} = (\mathsf{acc}.\mathbb{x}, \mathsf{acc}.\mathbb{w})$. As in Appendix C.3, oracle access includes the independent Fiat–Shamir interfaces when present, and query budgets count queries to all interfaces.

**Definition 8.1.** $\mathsf{ACC}$ is (perfectly) complete if, for every random-oracle output size $\sigma$, index $\mathbb{i}$, instance-witness pairs $(\mathbb{x}_i, \mathbb{w}_i)_{i \in [\ell_{\mathsf{acc},1}]}$, and accumulators $(\mathsf{acc}_j)_{j \in [\ell_{\mathsf{acc},2}]}$, the following experiment

$$
\begin{aligned}
& h \xleftarrow{\$} \mathcal{U}(\sigma) \\
& (\mathsf{pk}_{\mathsf{ACC}}, \mathsf{vk}_{\mathsf{ACC}}) \leftarrow \mathbf{I}^h_{\mathsf{ACC}}(\mathbb{i}) \\
& (\mathsf{acc}, \mathsf{pf}) \leftarrow \mathbf{P}^h_{\mathsf{ACC}}\left( \begin{array}{c} \mathsf{pk}_{\mathsf{ACC}}, (\mathbb{x}_i, \mathbb{w}_i)_{i \in [\ell_{\mathsf{acc},1}]}, \\ (\mathsf{acc}_j)_{j \in [\ell_{\mathsf{acc},2}]} \end{array} \right)
\end{aligned}
$$

satisfies

$$
\Pr\left[ \begin{array}{l} \quad \forall i \in [\ell_{\mathsf{acc},1}] : (\mathbb{i}, \mathbb{x}_i, \mathbb{w}_i) \in \mathcal{R} \\ \wedge \ \forall j \in [\ell_{\mathsf{acc},2}] : \mathbf{D}^h_{\mathsf{ACC}}(\mathbb{i}, \mathsf{acc}_j) = 1 \\ \quad \Downarrow \\ \quad \mathbf{V}^h_{\mathsf{ACC}}\left( \begin{array}{c} \mathsf{vk}_{\mathsf{ACC}}, (\mathbb{x}_i)_{i \in [\ell_{\mathsf{acc},1}]}, \\ (\mathsf{acc}_j.\mathbb{x})_{j \in [\ell_{\mathsf{acc},2}]}, \\ \mathsf{acc}.\mathbb{x}, \mathsf{pf} \end{array} \right) = 1 \\ \wedge \ \mathbf{D}^h_{\mathsf{ACC}}(\mathbb{i}, \mathsf{acc}) = 1 \end{array} \right] = 1 \ .
$$

**Definition 8.2.** $\mathsf{ACC}$ has straight-line knowledge soundness error $\kappa_{\mathsf{ACC}}$ with extraction time $\mathsf{et}_{\mathsf{ACC}}$ if there exists a relaxed decider $\widehat{\mathbf{D}}_{\mathsf{ACC}}$ (i.e., $\mathbf{D}_{\mathsf{ACC}}$ accepting implies $\widehat{\mathbf{D}}_{\mathsf{ACC}}$ accepting) and deterministic extractor $\mathbf{E}_{\mathsf{ACC}}$ such that, for every random oracle output size $\sigma$, index size bound $n_{\mathbb{i}}$, instance size bound $n_{\mathbb{x}}$, accumulator size bound $n_{\mathsf{acc}}$, number of instance-witness pairs $\ell_{\mathsf{acc},1}$, number of accumulators $\ell_{\mathsf{acc},2}$, and stateful adversary $\widetilde{\mathbf{P}}$ that performs at most $q$ queries to the random oracle, $\mathbf{E}_{\mathsf{ACC}}$ runs in time at most

$\mathsf{et}_{\mathsf{ACC}}(\sigma, q, n_{\mathbb{i}}, n_{\mathbb{x}}, n_{\mathsf{acc}}, \ell_{\mathsf{acc},1}, \ell_{\mathsf{acc},2})$ and the following experiment

$$
\begin{aligned}
&h \xleftarrow{\$} \mathcal{U}(\sigma)\\
&\mathbb{i} \xleftarrow{\mathsf{qtr}_1} \widetilde{\mathbf{P}}^h\\
&(\mathsf{pk}_{\mathsf{ACC}}, \mathsf{vk}_{\mathsf{ACC}}) \xleftarrow{\mathsf{qtr}_{\mathbf{I}_{\mathsf{ACC}}}} \mathbf{I}^h_{\mathsf{ACC}}(\mathbb{i})\\
&\begin{pmatrix} (\mathbb{x}_i)_{i\in[\ell_{\mathsf{acc},1}]},\\ (\mathsf{acc}_j.\mathbb{x})_{j\in[\ell_{\mathsf{acc},2}]},\\ \mathsf{acc}, \mathsf{pf} \end{pmatrix} \xleftarrow{\mathsf{qtr}_2} \widetilde{\mathbf{P}}^h(\mathsf{pk}_{\mathsf{ACC}}, \mathsf{vk}_{\mathsf{ACC}})\\
&b \xleftarrow{\mathsf{qtr}_{\mathbf{V}_{\mathsf{ACC}}}} \mathbf{V}^h_{\mathsf{ACC}}\begin{pmatrix} \mathsf{vk}_{\mathsf{ACC}}, (\mathbb{x}_i)_{i\in[\ell_{\mathsf{acc},1}]},\\ (\mathsf{acc}_j.\mathbb{x})_{j\in[\ell_{\mathsf{acc},2}]}, \mathsf{acc}.\mathbb{x}, \mathsf{pf} \end{pmatrix}\\
&\begin{pmatrix} (\mathbb{w}_i)_{i\in[\ell_{\mathsf{acc},1}]},\\ (\mathsf{acc}_j.\mathbb{w})_{j\in[\ell_{\mathsf{acc},2}]} \end{pmatrix} \leftarrow \mathbf{E}_{\mathsf{ACC}}\begin{pmatrix} \mathbb{i}, \mathsf{pk}_{\mathsf{ACC}}, \mathsf{vk}_{\mathsf{ACC}},\\ (\mathbb{x}_i)_{i\in[\ell_{\mathsf{acc},1}]}, (\mathsf{acc}_j.\mathbb{x})_{j\in[\ell_{\mathsf{acc},2}]},\\ \mathsf{acc}, \mathsf{pf}, \mathsf{qtr}_1, \mathsf{qtr}_2, \mathsf{qtr}_{\mathbf{I}_{\mathsf{ACC}}}, \mathsf{qtr}_{\mathbf{V}_{\mathsf{ACC}}} \end{pmatrix}
\end{aligned}
$$

has failure probability

$$
\begin{aligned}
\Pr\left[\begin{array}{l} |\mathbb{i}| \le n_{\mathbb{i}}\\ \wedge\ \forall i \in [\ell_{\mathsf{acc},1}] : |\mathbb{x}_i| \le n_{\mathbb{x}}\\ \wedge\ \forall j \in [\ell_{\mathsf{acc},2}] : |\mathsf{acc}_j.\mathbb{x}| \le n_{\mathsf{acc}}\\ \wedge\ b = 1\\ \wedge\ \widehat{\mathbf{D}}^h_{\mathsf{ACC}}(\mathbb{i}, \mathsf{acc}) = 1\\ \wedge\ \begin{pmatrix} \exists i \in [\ell_{\mathsf{acc},1}] : (\mathbb{i}, \mathbb{x}_i, \mathbb{w}_i) \notin \mathcal{R}\\ \vee\ \exists j \in [\ell_{\mathsf{acc},2}] : \widehat{\mathbf{D}}^h_{\mathsf{ACC}}(\mathbb{i}, \mathsf{acc}_j) \neq 1 \end{pmatrix} \end{array}\right]\\
\le \kappa_{\mathsf{ACC}}(\sigma, q, n_{\mathbb{i}}, n_{\mathbb{x}}, n_{\mathsf{acc}}, \ell_{\mathsf{acc},1}, \ell_{\mathsf{acc},2}) \ .
\end{aligned}
$$

The extractor makes no random-oracle queries; its trace inputs and trace-processing costs follow the conventions of Appendix C and Remark 3.18.

**Remark 8.3.** We note that the foregoing definition of accumulation schemes is from [BCFW25], and does not intrinsically support succinct deciders. There is a notion of accumulation schemes in the ROM from [BMNW25] that does support succinct deciders by associating the accumulator with a SNARK (by essentially providing the decider with a SNARK proof). For simplicity, we do not consider succinct deciders in this work, but note that our construction can straightforwardly be extended to support them.

**Properties of an accumulation scheme.** Some of the most important metrics of an accumulation scheme are its prover, verifier, and decider costs, which we report in terms of the number of field operations performed and the number of queries made to the random oracle. Additionally, for readability, we suppress the explicit dependence on parameters when reporting knowledge soundness errors and extraction times throughout the rest of this section.

## 8.2 Interactive accumulation schemes for $\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}}$

As one of the building blocks of our Canopy-based accumulation schemes, we require an interactive reduction that at a high level acts as an 'interactive accumulation scheme' for $\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}}$. Here MT abbreviates

$\mathsf{MT}[\sigma, \mathbb{F}, b]$, the top Merkle scheme of Canopy, with each commitment retaining its random-oracle interface when passed between reductions. More formally, we assume the existence of the following.

**Lemma 8.4.** *Let $\lambda$ be a target security parameter, $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, b, k, \delta_{\mathcal{C}})$, and let $\delta \in (0, \delta_{\mathsf{mca}}(\mathcal{C}))$.*[7] *There exists an indexed promise relation relative to a random oracle, denoted by $\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}}$, such that for all $\ell_{\mathsf{acc},1}, \ell_{\mathsf{acc},2} \in \mathbb{N}$, there exists an RBRKS interactive reduction $\mathsf{IACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}$ from $(\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{acc},1}} \times (\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}})^{\ell_{\mathsf{acc},2}}$ to $\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}}$ with the following properties:*

- *The RBRKS soundness error for each round is upper bounded by $\kappa^{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}_{\mathsf{IACC}}$.*
- *Prover cost: $\mathsf{pt}^{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}_{\mathsf{IACC}}$ field operations and $\mathsf{pq}^{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}_{\mathsf{IACC}}$ queries to the random oracle.*
- *Verifier cost: $\mathsf{vt}^{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}_{\mathsf{IACC}}$ field operations and $\mathsf{vq}^{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}_{\mathsf{IACC}}$ queries to the random oracle.*

*Furthermore, all tuples in $\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}}$ contain the same index $\mathbb{i}_{\mathsf{acc}}$, and checking strict membership of a purported tuple $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ in $\mathcal{R}^{h}_{\mathsf{acc}}$ can be decided using $\mathsf{dt}_{\mathsf{IACC}}$ field operations and $\mathsf{dq}_{\mathsf{IACC}}$ queries to the random oracle. Membership in the relaxed relation is also polynomial-time decidable.*

Such an interactive reduction for Reed–Solomon codes can be quite straightforwardly constructed using the techniques in WHIR [ACFY25], which can be used to construct an RBRKS IOR from $\mathsf{SEP}[\mathsf{RS}, \delta]^{\ell_{\mathsf{acc}}}$ to $\mathsf{SEP}[\mathsf{RS}, \delta]$, for all $\ell_{\mathsf{acc}} \in \mathbb{N}$ and all Reed–Solomon codes RS with smooth evaluation domains (i.e. the evaluation domain is a multiplicative coset in $\mathbb{F}^*$ of power-of-two size, and the message length is a power of two).[8] Importantly, this IOR has knowledge soundness errors $\leq \mathsf{negl}(\lambda)$ and makes $O\left(\frac{\lambda}{-\log(1-\delta)}\right)$ queries to each of the $\ell_{\mathsf{acc}}$ input oracles. The desired interactive accumulation scheme IACC for a given Reed–Solomon code RS can be obtained as follows: define $\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}} := \mathsf{CR}[\mathsf{SEP}[\mathsf{RS}, \delta], \mathsf{MT}]^{\mathcal{U}}$, the corresponding interactive reduction

$$\begin{aligned}\mathsf{IACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}} : (\mathsf{CR}[\mathsf{SEP}[\mathsf{RS}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{acc},1}} \times (\mathsf{CR}[\mathsf{SEP}[\mathsf{RS}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{acc},2}} \\ \to \mathsf{CR}[\mathsf{SEP}[\mathsf{RS}, \delta], \mathsf{MT}]^{\mathcal{U}}\end{aligned}$$

can be obtained by applying our transformation from Theorem 5.3 (setting all the VCRs to be Merkle trees) to the IOR described above, with $\ell_{\mathsf{acc}} := \ell_{\mathsf{acc},1} + \ell_{\mathsf{acc},2}$.[9]

This approach can be extended to other codes by using appropriate codeswitching IORs, such as those studied in [BCF+25; BCFW25; BMMS26a; BFRW26]. Importantly, none of the known codeswitching techniques for non-Reed–Solomon codes directly yield an IOR from $\mathsf{SEP}[\mathcal{C}, \delta]^{\ell_{\mathsf{acc}}}$ to $\mathsf{SEP}[\mathcal{C}, \delta]$ (especially in the list decoding setting where $\delta > \delta_{\mathcal{C}}/2$), instead requiring the IOR to output additional claims (involving, e.g., the generator matrix of the code [BMMS26a] or the right-extension of the code [BFRW26]). Therefore, for non-Reed–Solomon codes, we would not be able to simply set $\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}} := \mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}}$, as $\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}}$ will have to include such additional claims as well.

[7] We formally define $\delta_{\mathsf{mca}}(\mathcal{C})$, the proximity radius for mutual correlated agreement of the code $\mathcal{C}$, in Definition D.4. For affine-space generators, known results give $\delta_{\mathsf{mca}}(\mathcal{C}) \geq 1 - (1 - \delta_{\mathcal{C}} + \epsilon)^{1/3}$ for $\epsilon \in (0, \delta_{\mathcal{C}})$, when the field is sufficiently large relative to the other parameters.

[8] As described in [ACFY25, Section 2.1.3], the WHIR IOPP is constructed by composing a sequence of IORs, and underlying each IOR is a 'codeswitching' IOR from $\mathsf{SEP}[\mathsf{RS}, \delta]$ to $\mathsf{SEP}[\mathsf{RS}', \delta]$, for arbitrary Reed–Solomon codes RS and RS′ with smooth evaluation domains (importantly, we can set $\mathsf{RS} = \mathsf{RS}'$). Our desired IOR from $\mathsf{SEP}[\mathsf{RS}, \delta]^{\ell_{\mathsf{acc}}}$ to $\mathsf{SEP}[\mathsf{RS}, \delta]$ can be obtained as follows: (i) use the techniques from Appendix D to reduce the task of checking the $\ell_{\mathsf{acc}}$ $\mathsf{SEP}[\mathsf{RS}, \delta]$ claims to checking a single 'virtual' $\mathsf{MEP}[\mathsf{RS}, \delta]$ claim, viewed as a special case of $\mathsf{SEP}[\mathsf{RS}, \delta]$ (this 'virtual' oracle is a random linear combination of the input oracles, whose queries are simulated by querying and combining the inputs; using independent uniform coefficients for codeword batching gives the affine-space mutual-correlated-agreement bound above), and (ii) use the WHIR codeswitch IOR from $\mathsf{SEP}[\mathsf{RS}, \delta]$ to $\mathsf{SEP}[\mathsf{RS}, \delta]$ to reduce checking this virtual SEP claim to checking a single 'actual' output $\mathsf{SEP}[\mathsf{RS}, \delta]$ claim.

[9] Since Merkle commitments have an empty bad relation, the bad-certificate branches are empty and the good tags can be omitted. The compiled input relation agrees with the displayed product after regrouping.

### 8.3 Canopy-based accumulation scheme construction

In order to construct a Canopy-based accumulation scheme in the ROM for an indexed relation $\mathcal{R}$, we expect as input (as in prior work [BMNW25; BMMS26a]) an indexed oracle relation $\mathcal{R}_{\mathsf{cast}}$ and two kinds of IORs:

- $\mathsf{CAST}$, an RBRKS IOR from $\mathcal{R}$ to $\mathcal{R}_{\mathsf{cast}}$.
- For all $\ell_{\mathsf{acc}} \in \mathbb{N}$, an RBRKS IOR $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$ from $\mathcal{R}_{\mathsf{cast}}^{\ell_{\mathsf{acc}}}$ to $\mathcal{R}_{\mathsf{cast}}$.

Prior work [BMNW25; BCFW25; BMMS26a] presents instantiations for such $\mathsf{CAST}$ and $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$ IORs, by making use of error-correcting codes. We make the following assumptions regarding these IORs:[10]

- Every tuple in $\mathcal{R}_{\mathsf{cast}}$ has $\ell_{\mathsf{cast}} = O(1)$ many instance oracles, all of equal length $\mathsf{len}_{\mathsf{cast}} = O(n)$.
- Both IORs have empty index oracles. The verifier of $\mathsf{CAST}$ makes no queries to any oracles, and every nonempty proof oracle of $\mathsf{CAST}$ is selected as an output oracle.
- Apart from its output claim, the $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$ IOR involves the verifier querying $\ell_{\mathsf{fold}} = O(\ell_{\mathsf{acc}})$ many oracles at $O(\lambda)$ indices each, where all the oracles are of length $\mathsf{len}_{\mathsf{cast}}$.[11]

We present the full construction of the Canopy-based accumulation scheme in Fig. 5 and discuss its properties in Theorem 8.5.

**Theorem 8.5.** *Consider the following ingredients:*

- *An RBRKS IOR* $\mathsf{CAST}$ *from* $\mathcal{R}$ *to* $\mathcal{R}_{\mathsf{cast}}$.
- *For all* $\ell_{\mathsf{acc}} \in \mathbb{N}$*, an RBRKS IOR* $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$ *from* $\mathcal{R}_{\mathsf{cast}}^{\ell_{\mathsf{acc}}}$ *to* $\mathcal{R}_{\mathsf{cast}}$.
- *A linear code* $\mathcal{C}$ *with parameters* $(\mathbb{F}, b, k, \delta_{\mathcal{C}})$ *and a corresponding proximity parameter* $\delta \in (0, \delta_{\mathsf{mca}}(\mathcal{C}))$.
- *For all* $\ell_{\mathsf{acc},1}, \ell_{\mathsf{acc},2} \in \mathbb{N}$*, an RBRKS IR* $\mathsf{IACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}$ *from*

$$(\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{acc},1}} \times (\mathcal{R}_{\mathsf{acc}}^{\mathcal{U}})^{\ell_{\mathsf{acc},2}}$$

*to* $\mathcal{R}_{\mathsf{acc}}^{\mathcal{U}}$ *as described in Lemma 8.4.*

*Let all the ingredients be perfectly complete, satisfy the hypotheses of Theorems 5.3 and 5.4, and satisfy the simplifying assumptions stated in Fig. 5. For all* $\ell_{\mathsf{acc},1}, \ell_{\mathsf{acc},2} \in \mathbb{N}$ *with* $\ell_{\mathsf{acc},1} + \ell_{\mathsf{acc},2} \geq 1$*, the construction* $\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}$ *in Fig. 5 is an accumulation scheme in the random oracle model that accumulates* $\ell_{\mathsf{acc},1}$ *statements and* $\ell_{\mathsf{acc},2}$ *accumulators, with perfect completeness and straight-line knowledge soundness with the error and extraction-time bounds of Section 8.3.1.*

The perfect completeness and straight-line knowledge soundness claims of Theorem 8.5 follow quite straightforwardly from the guarantees of the ingredients used, and the results discussed in the rest of the paper. We provide an intuitive description of how these results are applied below. In the interest of clarity, we gloss over a technical subtlety which we discuss separately in Remark 8.6.

- The constituent IRs are perfectly complete and RBRKS as discussed below:
  - The fact that $\mathsf{IACC}_{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$ is a perfectly complete and RBRKS IR from $(\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{fold}}} \times (\mathcal{R}_{\mathsf{acc}}^{\mathcal{U}})^{\ell_{\mathsf{acc},2}}$ to $\mathcal{R}_{\mathsf{acc}}^{\mathcal{U}}$ follows from Lemma 8.4.
  - The fact that $\mathsf{ICAST}$ is a perfectly complete and RBRKS IR from $\mathcal{R}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}}$ follows from Theorem 5.4 and Section 6.
  - We fact that $\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}$ is a perfectly complete and RBRKS IR from $(\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}})^{\ell_{\mathsf{acc}}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times (\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{fold}}}$. This follows from Theorem 5.4, regrouping the committed input witnesses as in Definition 5.1.

[10] All of the simplifying assumptions are natural properties satisfied by the existing instantiations from [BMNW25; BCFW25; BMMS26a].

[11] All the oracles in the $\mathsf{CAST}$ and $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$ IORs are of the same length as they are all purported codewords of the code underlying the construction of these IORs.

- It then follows from Lemma C.5 that $\mathsf{IR}_{\mathsf{acc}}$ is a perfectly complete and RBRKS IR from $\mathcal{R}^{\ell_{\mathsf{acc},1}} \times (\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}_{\mathsf{acc}}^{\mathcal{U}})^{\ell_{\mathsf{acc},2}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}_{\mathsf{acc}}^{\mathcal{U}}$, since it is obtained by sequentially composing $\ell_{\mathsf{acc},1}$ instances of $\mathsf{ICAST}$, $\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}$, and $\mathsf{IACC}_{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$.
- Finally, Lemmas C.4 and C.7 give that $\mathsf{RDX}_{\mathsf{acc}}$ is a perfectly complete and straight-line knowledge sound non-interactive reduction from $\mathcal{R}^{\ell_{\mathsf{acc},1}} \times (\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}_{\mathsf{acc}}^{\mathcal{U}})^{\ell_{\mathsf{acc},2}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}_{\mathsf{acc}}^{\mathcal{U}}$, since it is obtained by applying the hash-chain Fiat–Shamir transformation to the RBRKS IR $\mathsf{IR}_{\mathsf{acc}}$.
- The perfect completeness and straight-line knowledge soundness of $\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}$ follow from the analogous guarantees of $\mathsf{RDX}_{\mathsf{acc}}$. We provide precise parameter bookkeeping in Section 8.3.1.

**Remark 8.6** (Applying Theorem 5.4 to $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$)**.** In the foregoing discussion and in Fig. 5, in the interest of clarity, we say that Theorem 5.4 gives that $\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}$ is a perfectly complete and RBRKS IR from $(\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}})^{\ell_{\mathsf{acc}}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times (\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{fold}}}$. However, in order to apply Theorem 5.4 to $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$, we must provide non-trivial upper bounds $\beta_\tau$ for each input oracle string $\tau \in S_0$. As discussed in Section 5.4, in order to provide such upper bounds for IRs where the input VCR commitments can be arbitrarily chosen by the adversary,[12] we must enforce that the challenges in the input Canopy commitments must be consistent with the Fiat–Shamir transform (i.e., must be obtained by hashing the transcript of the commitment interaction so far). This can be implemented via a minor modification (which is more syntactic than technical), wherein we define $\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}$ to be an IR from $(\mathcal{R}_{\mathsf{FS}}^{\mathcal{U}})^{\ell_{\mathsf{acc}}}$ to $\mathcal{R}_{\mathsf{FS}}^{\mathcal{U}} \times (\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{fold}}}$ instead, where $\mathcal{R}_{\mathsf{FS}}$ is defined analogously to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]$, but additionally enforces the foregoing Fiat–Shamir challenge consistency. In the interest of clarity (and because in practice, these commitments will not actually be adversarially chosen, but will instead be the output of earlier honest executions of the accumulation scheme), we omit this largely syntactic modification in Fig. 5.

**Input ingredients:**

- The number of statements and accumulators being accumulated, denoted by $\ell_{\mathsf{acc},1}$ and $\ell_{\mathsf{acc},2}$, respectively. We define $\ell_{\mathsf{acc}} := \ell_{\mathsf{acc},1} + \ell_{\mathsf{acc},2}$.
- An indexed oracle relation $\mathcal{R}_{\mathsf{cast}}$, with every tuple having $\ell_{\mathsf{cast}}$ many instance oracles of length $\mathsf{len}_{\mathsf{cast}}$.
- An IOR $\mathsf{CAST}$ from $\mathcal{R}$ to $\mathcal{R}_{\mathsf{cast}}$, with no queried oracles.
- An IOR $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$ from $\mathcal{R}_{\mathsf{cast}}^{\ell_{\mathsf{acc}}}$ to $\mathcal{R}_{\mathsf{cast}}$, where the verifier queries $\ell_{\mathsf{fold}}$ many oracles of length $\mathsf{len}_{\mathsf{cast}}$, including every nonempty oracle not selected as output.
- A code $\mathcal{C}$ with parameters $(\mathbb{F}, b, k, \delta_{\mathcal{C}})$ and a proximity parameter $\delta \in (0, \delta_{\mathsf{mca}}(\mathcal{C}))$.
- An indexed relation with respect to a random oracle $\mathcal{R}_{\mathsf{acc}}^{\mathcal{U}}$.
- An interactive reduction $\mathsf{IACC}_{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$ from $(\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{fold}}} \times (\mathcal{R}_{\mathsf{acc}}^{\mathcal{U}})^{\ell_{\mathsf{acc},2}}$ to $\mathcal{R}_{\mathsf{acc}}^{\mathcal{U}}$.

**Defining the underlying non-interactive reduction:**

Define the non-interactive reduction (see Appendix C.3) underlying the accumulation scheme as follows:

- Define $\mathsf{Canopy} := \mathsf{Canopy}[\sigma, \mathbb{F}, \mathsf{len}_{\mathsf{cast}}, \mathcal{C}, \delta]$.
- Define $\vec{\mathsf{V}}_{\mathsf{all}}^{\mathsf{cast}}$ and $\vec{\mathsf{V}}_{\mathsf{all}}^{\mathsf{fold},\ell_{\mathsf{acc}}}$ to be the VCR assignments for $\mathsf{CAST}$ and $\mathsf{FOLD}$, respectively, that assign $\mathsf{Canopy}$ to all nonempty oracles and the trivial VCR scheme of Section 5.2 to empty oracles. Suppress trivial output factors and retain the random-oracle interfaces of existing commitments.

[12] $\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}$ turns out to be such an IR, since Definition 8.2 implies that the accumulation adversary can adversarially chose the input accumulators $(\mathsf{acc}_j.\mathbb{x})_{j \in [\ell_{\mathsf{acc},2}]}$, which contain Canopy VCR commitments later given as input to $\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}$.

- Apply Theorem 5.4 to define the following interactive reductions:
  - $\mathsf{ICAST} := \mathsf{T}[\mathsf{CAST}, \vec{\mathsf{V}}^{\mathsf{cast}}_{\mathsf{all}}]$, from $\mathcal{R}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}}$.
  - $\mathsf{IFOLD}_{\ell_{\mathsf{acc}}} := \mathsf{T}[\mathsf{FOLD}_{\ell_{\mathsf{acc}}}, \vec{\mathsf{V}}^{\mathsf{fold},\ell_{\mathsf{acc}}}_{\mathsf{all}}]$, from $(\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}})^{\ell_{\mathsf{acc}}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times (\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{fold}}}$.
- Define the interactive reduction $\mathsf{IR}_{\mathsf{acc}}$ from $\mathcal{R}^{\ell_{\mathsf{acc},1}} \times (\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}^{\mathcal{U}}_{\mathsf{acc}})^{\ell_{\mathsf{acc},2}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}^{\mathcal{U}}_{\mathsf{acc}}$ by composing the following IRs:
  - Use $\ell_{\mathsf{acc},1}$ sequential applications of $\mathsf{ICAST}$ to obtain $\mathsf{IR}_1$ from $\mathcal{R}^{\ell_{\mathsf{acc},1}} \times (\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}^{\mathcal{U}}_{\mathsf{acc}})^{\ell_{\mathsf{acc},2}}$ to $(\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}})^{\ell_{\mathsf{acc}}} \times (\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}})^{\ell_{\mathsf{acc},2}}$.
  - Compose $\mathsf{IR}_1$ with $\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}$ to obtain $\mathsf{IR}_2$ from $\mathcal{R}^{\ell_{\mathsf{acc},1}} \times (\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}^{\mathcal{U}}_{\mathsf{acc}})^{\ell_{\mathsf{acc},2}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times (\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{fold}}} \times (\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}})^{\ell_{\mathsf{acc},2}}$.
  - Compose $\mathsf{IR}_2$ with $\mathsf{IACC}_{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$ to obtain the desired interactive reduction $\mathsf{IR}_{\mathsf{acc}}$ from $\mathcal{R}^{\ell_{\mathsf{acc},1}} \times (\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}^{\mathcal{U}}_{\mathsf{acc}})^{\ell_{\mathsf{acc},2}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}^{\mathcal{U}}_{\mathsf{acc}}$.
- Define $\mathsf{RDX}_{\mathsf{acc}} = (\mathbf{I}_{\mathsf{NI}}, \mathbf{P}_{\mathsf{NI}}, \mathbf{V}_{\mathsf{NI}})$ to be the non-interactive reduction from $\mathcal{R}^{\ell_{\mathsf{acc},1}} \times (\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}^{\mathcal{U}}_{\mathsf{acc}})^{\ell_{\mathsf{acc},2}}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times \mathcal{R}^{\mathcal{U}}_{\mathsf{acc}}$ obtained by applying the hash-chain Fiat–Shamir transformation of Appendix C.3 to $\mathsf{IR}_{\mathsf{acc}}$.

**Description of the accumulation scheme:**
We describe the algorithms of $\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}} = (\mathbf{I}_{\mathsf{ACC}}, \mathbf{P}_{\mathsf{ACC}}, \mathbf{V}_{\mathsf{ACC}}, \mathbf{D}_{\mathsf{ACC}})$ and its relaxed decider below:

- $\mathbf{I}^h_{\mathsf{ACC}}(\mathbb{i})$:
  1. Obtain $\mathbb{i}_{\mathsf{cast}}$ from the indexer of $\mathsf{ICAST}$ on $\mathbb{i}$, and define $\mathbb{i}_{\mathsf{NI}} := (\mathbb{i}^{\ell_{\mathsf{acc},1}}, (\mathbb{i}_{\mathsf{cast}}, \mathbb{i}_{\mathsf{acc}})^{\ell_{\mathsf{acc},2}})$.
  2. Compute $(\mathbb{i}'_{\mathsf{NI}}, \mathsf{pk}_{\mathsf{NI}}, \mathsf{vk}_{\mathsf{NI}}) \leftarrow \mathbf{I}^h_{\mathsf{NI}}(\mathbb{i}_{\mathsf{NI}})$.
  3. Output $(\mathsf{pk}_{\mathsf{ACC}}, \mathsf{vk}_{\mathsf{ACC}}) := (\mathsf{pk}_{\mathsf{NI}}, \mathsf{vk}_{\mathsf{NI}})$.
- $\mathbf{P}^h_{\mathsf{ACC}}(\mathsf{pk}_{\mathsf{ACC}}, (\mathbb{x}_i, \mathbb{w}_i)_{i\in[\ell_{\mathsf{acc},1}]}, (\mathsf{acc}_j)_{j\in[\ell_{\mathsf{acc},2}]})$:
  1. Parse $\mathsf{pk}_{\mathsf{ACC}}$ as $\mathsf{pk}_{\mathsf{NI}}$.
  2. Define $\mathbb{x}_{\mathsf{NI}} := ((\mathbb{x}_i)_{i\in[\ell_{\mathsf{acc},1}]}, (\mathsf{acc}_j.\mathbb{x})_{j\in[\ell_{\mathsf{acc},2}]})$.
  3. Compute
$$(\mathsf{pf}, \mathbb{x}'_{\mathsf{NI}}, \mathbb{w}'_{\mathsf{NI}}) \leftarrow \mathbf{P}^h_{\mathsf{NI}}\left(\begin{array}{c}\mathsf{pk}_{\mathsf{NI}}, \mathbb{x}_{\mathsf{NI}},\\ ((\mathbb{w}_i)_{i\in[\ell_{\mathsf{acc},1}]}, (\mathsf{acc}_j.\mathbb{w})_{j\in[\ell_{\mathsf{acc},2}]})\end{array}\right).$$
  4. Define $\mathsf{acc}.\mathbb{x} := \mathbb{x}'_{\mathsf{NI}}$ and $\mathsf{acc}.\mathbb{w} := \mathbb{w}'_{\mathsf{NI}}$.
  5. Output $(\mathsf{acc}, \mathsf{pf})$.
- $\mathbf{V}^h_{\mathsf{ACC}}(\mathsf{vk}_{\mathsf{ACC}}, (\mathbb{x}_i)_{i\in[\ell_{\mathsf{acc},1}]}, (\mathsf{acc}_j.\mathbb{x})_{j\in[\ell_{\mathsf{acc},2}]}, \mathsf{acc}.\mathbb{x}, \mathsf{pf})$:
  1. Parse $\mathsf{vk}_{\mathsf{ACC}}$ as $\mathsf{vk}_{\mathsf{NI}}$.
  2. Define $\mathbb{x}_{\mathsf{NI}} := ((\mathbb{x}_i)_{i\in[\ell_{\mathsf{acc},1}]}, (\mathsf{acc}_j.\mathbb{x})_{j\in[\ell_{\mathsf{acc},2}]})$.
  3. Compute $(b, \mathbb{x}'_{\mathsf{NI}}) \leftarrow \mathbf{V}^h_{\mathsf{NI}}(\mathsf{vk}_{\mathsf{NI}}, \mathbb{x}_{\mathsf{NI}}, \mathsf{pf})$.
  4. Output 1 if and only if $b = 1$ and $\mathbb{x}'_{\mathsf{NI}} = \mathsf{acc}.\mathbb{x}$.
- $\mathbf{D}^h_{\mathsf{ACC}}(\mathbb{i}, \mathsf{acc})$:

1. Parse acc as $(\mathbb{x}'_{\mathsf{NI}}, \mathbb{w}'_{\mathsf{NI}})$, and obtain $\mathbb{i}_{\mathsf{cast}}$ by running the indexer of $\mathsf{ICAST}$ on $\mathbb{i}$.
2. Output 1 if and only if $((\mathbb{i}_{\mathsf{cast}}, \mathbb{i}_{\mathsf{acc}}), \mathbb{x}'_{\mathsf{NI}}, \mathbb{w}'_{\mathsf{NI}}) \in \mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^h \times \mathcal{R}^h_{\mathsf{acc}}$ (strict membership).

- $\widehat{\mathbf{D}}^h_{\mathsf{ACC}}(\mathbb{i}, \mathsf{acc})$:
  1. Parse acc and obtain $\mathbb{i}_{\mathsf{cast}}$ as in $\mathbf{D}_{\mathsf{ACC}}$.
  2. Output 1 if and only if $((\mathbb{i}_{\mathsf{cast}}, \mathbb{i}_{\mathsf{acc}}), \mathbb{x}'_{\mathsf{NI}}, \mathbb{w}'_{\mathsf{NI}}) \in \widetilde{\mathsf{CR}}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^h \times \tilde{\mathcal{R}}^h_{\mathsf{acc}}$ (relaxed membership).

**Figure 5:** The Canopy-based accumulation scheme for $\mathcal{R}$ discussed in Theorem 8.5.

**Remark 8.7** (Shared random-oracle interfaces)**.** The compiler setup in Section 5.2 assigns distinct random-oracle tags to distinct oracle labels. In Fig. 5, we allow different labels to share an interface, with the same output length, so that commitments retain their interfaces across compositions. This deviation from the compiler setup does not weaken the guarantees of Theorems 5.3 and 5.4 or worsen their bounds. The proof in Appendix A can be naturally extended to this setting by providing each component extractor with the entire query trace projected to its interface, including queries associated with other commitments sharing that interface. In each component security reduction, all calls to the shared interface are forwarded to the component's random oracle. The existing query bounds account for these calls, and the error analysis uses union bounds without requiring independence between components.

### 8.3.1 Knowledge soundness and efficiency metrics

Before we can describe the knowledge soundness and efficiency metrics of $\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}$ from Theorem 8.5, we begin by establishing some notation for the metrics for the underlying ingredients discussed in Fig. 5:

- Let the RBRKS IR $\mathsf{IACC}_{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$ from $(\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{fold}}} \times (\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}})^{\ell_{\mathsf{acc},2}}$ to $\mathcal{R}^{\mathcal{U}}_{\mathsf{acc}}$ have the following metrics:
  - $\mathsf{k}_{\mathsf{IACC}}^{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$ rounds.
  - RBRKS soundness errors
    $$[\kappa_i^{\mathsf{IACC}_{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}}]_{i\in[\mathsf{k}_{\mathsf{IACC}}^{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}]}$$
    and extraction times
    $$[\mathsf{et}_i^{\mathsf{IACC}_{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}}]_{i\in[\mathsf{k}_{\mathsf{IACC}}^{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}]}.$$
  - Prover costs: $\mathsf{pt}_{\mathsf{IACC}}^{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$ field operations and $\mathsf{pq}_{\mathsf{IACC}}^{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$ queries to the random oracle.
  - Verifier costs: $\mathsf{vt}_{\mathsf{IACC}}^{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$ field operations and $\mathsf{vq}_{\mathsf{IACC}}^{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$ queries to the random oracle.
- Let the RBRKS IR $\mathsf{ICAST} := \mathsf{T}[\mathsf{CAST}, \vec{\mathsf{V}}^{\mathsf{cast}}_{\mathsf{all}}]$, from $\mathcal{R}$ to $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}}$ have the following metrics:
  - $\mathsf{k}_{\mathsf{ICAST}}$ rounds.
  - RBRKS soundness errors $[\kappa_i^{\mathsf{ICAST}}]_{i\in[\mathsf{k}_{\mathsf{ICAST}}]}$ and extraction times $[\mathsf{et}_i^{\mathsf{ICAST}}]_{i\in[\mathsf{k}_{\mathsf{ICAST}}]}$.
  - Prover costs: $\mathsf{pt}_{\mathsf{ICAST}}$ field operations and $\mathsf{pq}_{\mathsf{ICAST}}$ queries to the random oracle.

– Verifier costs: $\mathsf{vt}_{\mathsf{ICAST}}$ field operations and $\mathsf{vq}_{\mathsf{ICAST}}$ queries to the random oracle.

- Let the RBRKS IR
$$\mathsf{IFOLD}_{\ell_{\mathsf{acc}}} := \mathsf{T}[\mathsf{FOLD}_{\ell_{\mathsf{acc}}}, \vec{\mathsf{V}}_{\mathsf{all}}^{\mathsf{fold},\ell_{\mathsf{acc}}}]$$
from $(\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}})^{\ell_{\mathsf{acc}}}$ to
$$\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^{\mathcal{U}} \times (\mathsf{CR}[\mathsf{SEP}[\mathcal{C}, \delta], \mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{fold}}}$$
have the following metrics:
  – $\mathsf{k}_{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}$ rounds.
  – RBRKS soundness errors
$$[\kappa_i^{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}]_{i\in[\mathsf{k}_{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}]}$$
and extraction times
$$[\mathsf{et}_i^{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}]_{i\in[\mathsf{k}_{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}]} .$$
  – Prover costs: $\mathsf{pt}_{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}$ field operations and $\mathsf{pq}_{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}$ queries to the random oracle.
  – Verifier costs: $\mathsf{vt}_{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}$ field operations and $\mathsf{vq}_{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}$ queries to the random oracle.

Note that the metrics of $\mathsf{ICAST}$ and $\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}$ (given by Theorem 5.4) already take into account the metrics of the underlying VCR schemes. The upper bounds $\beta_\tau$ for the Canopy commitments used can be derived as discussed in Section 5.4 and Remark 8.6.

**The metrics of $\mathsf{IR}_{\mathsf{acc}}$.** Set $\ell_{\mathsf{acc}} := \ell_{\mathsf{acc},1} + \ell_{\mathsf{acc},2}$, and abbreviate the round complexity notation as follows
$$k_{\mathsf{cast}} := \mathsf{k}_{\mathsf{ICAST}}, \qquad k_{\mathsf{fold}} := \mathsf{k}_{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}, \qquad k_{\mathsf{IACC}} := \mathsf{k}_{\mathsf{IACC}}^{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}.$$
The total number of rounds of $\mathsf{IR}_{\mathsf{acc}}$ is
$$k_{\mathsf{acc}} := \ell_{\mathsf{acc},1} k_{\mathsf{cast}} + k_{\mathsf{fold}} + k_{\mathsf{IACC}},$$
since the $\ell_{\mathsf{acc},1}$ casting reductions run sequentially. If $\ell_{\mathsf{acc},1} = 0$, the casting stage is empty.

As in Theorem 7.2, evaluate component bounds with the statement-size and query-budget substitutions of Lemmas C.4 and C.5. For each $r \in [k_{\mathsf{acc}}]$ define
$$(\kappa_r, \mathsf{et}_r) := \begin{cases} (\kappa_i^{\mathsf{ICAST}}, \mathsf{et}_i^{\mathsf{ICAST}}), & r = (j-1)k_{\mathsf{cast}} + i, \quad j \in [\ell_{\mathsf{acc},1}],\ i \in [k_{\mathsf{cast}}], \\ (\kappa_i^{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}, \mathsf{et}_i^{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}}), & r = \ell_{\mathsf{acc},1} k_{\mathsf{cast}} + i, \quad i \in [k_{\mathsf{fold}}], \\ (\kappa_i^{\mathsf{IACC}_{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}}, \mathsf{et}_i^{\mathsf{IACC}_{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}}), & r = \ell_{\mathsf{acc},1} k_{\mathsf{cast}} + k_{\mathsf{fold}} + i, \quad i \in [k_{\mathsf{IACC}}]. \end{cases}$$
For the efficiency metrics, write
$$c_\Sigma := \ell_{\mathsf{acc},1} c_{\mathsf{ICAST}} + c_{\mathsf{IFOLD}_{\ell_{\mathsf{acc}}}} + c_{\mathsf{IACC}}^{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}} \qquad \text{for } c \in \{\mathsf{pt}, \mathsf{pq}, \mathsf{vt}, \mathsf{vq}\}.$$

**The metrics of the accumulation scheme.** In order to obtain $\mathsf{RDX}_{\mathsf{acc}}$, we apply the hash-chain Fiat–Shamir transformation to $\mathsf{IR}_{\mathsf{acc}}$, padding its challenges to length at least $\sigma$ as in Appendix C.3, and use the same parameter bookkeeping as in Theorem 7.2. Let $q$ bound the adversary's random-oracle queries in Definition 8.2. Write $q^* := q + q_{\mathbf{I}_{\mathsf{ACC}}}$, where $q_{\mathbf{I}_{\mathsf{ACC}}}$ bounds the indexer's queries: the adversary against the underlying non-interactive reduction simulates this computation to supply the accumulation keys. Using the metrics defined above, the knowledge soundness and efficiency bounds for $\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}$ are as follows:

- Straight-line knowledge soundness error:

$$\kappa_{\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}} \leq (q^*+1)\sum_{r\in[k_{\mathsf{acc}}]}\kappa_r + \frac{(q^*+k_{\mathsf{acc}}+1)^2}{2^\sigma}.$$

- Extraction time:

$$\mathsf{et}_{\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}} \leq \sum_{r\in[k_{\mathsf{acc}}]}\mathsf{et}_r + O\big((k_{\mathsf{acc}}+1)T_{\mathsf{IR}_{\mathsf{acc}}}\big)\ ,$$

  where $T_{\mathsf{IR}_{\mathsf{acc}}}$ bounds the total time of one execution of $\mathbf{I}_{\mathsf{ACC}}$ and the verifier of $\mathsf{IR}_{\mathsf{acc}}$; component bounds use adversarial query budget $q^*$ before the substitutions above.

- Prover costs, reusing verifier computations when forming intermediate and final instances:

$$\mathsf{pt}_{\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}} \leq \mathsf{pt}_\Sigma + \mathsf{vt}_\Sigma, \qquad \mathsf{pq}_{\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}} \leq \mathsf{pq}_\Sigma + \mathsf{vq}_\Sigma + k_{\mathsf{acc}}.$$

- Verifier costs, including one Fiat–Shamir query per round:

$$\mathsf{vt}_{\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}} \leq \mathsf{vt}_\Sigma, \qquad \mathsf{vq}_{\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}} \leq \mathsf{vq}_\Sigma + k_{\mathsf{acc}}.$$

- Decider costs: let $\mathsf{dt}_{\mathsf{CR}}$ and $\mathsf{dq}_{\mathsf{CR}}$ be the field-operation and random-oracle query costs of computing $\mathbb{i}_{\mathsf{cast}}$ and deciding strict membership in $\mathsf{CR}[\mathcal{R}_{\mathsf{cast}}, \mathsf{Canopy}^{\ell_{\mathsf{cast}}}]^h$. Then

$$\mathsf{dt}_{\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}} \leq \mathsf{dt}_{\mathsf{CR}} + \mathsf{dt}_{\mathsf{IACC}}, \qquad \mathsf{dq}_{\mathsf{ACC}_{\ell_{\mathsf{acc},1},\ell_{\mathsf{acc},2}}} \leq \mathsf{dq}_{\mathsf{CR}} + \mathsf{dq}_{\mathsf{IACC}}.$$

### 8.3.2 Instantiating the ingredients to obtain Theorem 1.3

The result in Theorem 1.3 follows from the ingredients below for fixed accumulation arities. Here $\lambda$ is the target security parameter and $n = \mathrm{poly}(\lambda)$ bounds the computation and witness size, with $n = \Omega(\lambda \log n)$. We take $\sigma = \Theta(\lambda)$, polynomially bounded adversarial query budgets, and a sufficiently large field. All knowledge soundness errors below are negligible in $\lambda$ and all extraction times are polynomial in $\lambda$.

**The CAST and FOLD$_{\ell_{\mathsf{acc}}}$ IORs.** For our CAST and $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$, we use the instantiations from [BCFW25; BMMS26a] (which leverage linear-time encodable codes),[13] which have the following properties:

- The CAST IOR has the following properties:
  - Prover cost: $O(n)$ field operations.
  - Verifier cost: $O(\log n)$ field operations and no queries to oracles.
- The $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$ IOR has the following properties:
  - Prover cost: $O(\ell_{\mathsf{acc}} \cdot n)$ field operations.
  - Verifier cost: $O(\ell_{\mathsf{acc}} \cdot \lambda \cdot \log n)$ field operations and $O(\ell_{\mathsf{acc}} \cdot \lambda)$ queries to oracles.
  - In particular, the verifier queries $\ell_{\mathsf{fold}} = O(\ell_{\mathsf{acc}})$ oracles of length $\mathsf{len}_{\mathsf{cast}} = O(n)$, each at $\mathsf{qsize}_{\mathsf{fold}} = O(\lambda)$ indices.

**The interactive accumulation scheme.** We use an RBRKS interactive reduction $\mathsf{IACC}_{\ell_{\mathsf{fold}},\ell_{\mathsf{acc},2}}$ from $(\mathsf{CR}[\mathsf{SEP}[\mathcal{C},\delta],\mathsf{MT}]^{\mathcal{U}})^{\ell_{\mathsf{fold}}} \times (\mathcal{R}_{\mathsf{acc}}^{\mathcal{U}})^{\ell_{\mathsf{acc},2}}$ to $\mathcal{R}_{\mathsf{acc}}^{\mathcal{U}}$ with the following properties:

- The RBRKS soundness error for each round is $\mathrm{negl}(\lambda)$.

[13] Strictly speaking, the CAST and $\mathsf{FOLD}_{\ell_{\mathsf{acc}}}$ instantiations from [BMMS26a] are not proven to have RBRKS, but it is straightforward to prove this using the proof techniques of [BCFW25].

- Prover cost:
  - $O((\ell_{\mathsf{fold}} + \ell_{\mathsf{acc},2}) \cdot b + T_{\mathcal{C}})$ field operations, where $T_{\mathcal{C}}$ and $b$ denote the encoding time and block length of $\mathcal{C}$, respectively, and
  - $O(b)$ queries to the random oracle.
- Verifier cost:
  - $O((\ell_{\mathsf{fold}} + \ell_{\mathsf{acc},2}) \cdot \log b \cdot \lambda)$ field operations, and
  - $O\left((\ell_{\mathsf{fold}} + \ell_{\mathsf{acc},2}) \cdot \left(\frac{\lambda}{-\log(1-\delta)}\right) \cdot \log b\right)$ queries to the random oracle.

As discussed in Section 8.2, such interactive accumulation schemes can be constructed for Reed–Solomon codes using techniques from [ACFY25], and for other codes by using the techniques in [BCF+25; BCFW25; BMMS26a; BFRW26]. We discuss our specific choice of code $\mathcal{C}$ next. For these instantiations we use $O(1)$ fresh nonempty oracles per cast or fold, $\mathsf{len}_{\mathsf{cast}} = \Theta(n)$, and $k_{\mathsf{acc}} = O(\ell_{\mathsf{acc}} \log n)$, including the sequential casting rounds. Strict membership in the final accumulator relation, including computing its index and evaluating its SEP claim, costs $O(n)$ field operations and random-oracle queries.

**Choice of code $\mathcal{C}$.** We are left to choose an appropriate code $\mathcal{C}$ with parameters $(\mathbb{F}, b, k, \delta_{\mathcal{C}})$ and encoding time $T_{\mathcal{C}}$ that strikes a balance between the following considerations:

- To ensure that the commitment and opening prover times of Canopy are $O(n)$, we require that $T_{\mathcal{C}} = O(n)$ and $b = O(n)$.
- We would like to minimize the verifier cost of the interactive accumulation scheme, which involves $O((\ell_{\mathsf{fold}} + \ell_{\mathsf{acc},2}) \cdot \log b \cdot \lambda) = O(\ell_{\mathsf{acc}} \cdot \log n \cdot \lambda)$ field operations and $O\left((\ell_{\mathsf{fold}} + \ell_{\mathsf{acc},2}) \cdot \left(\frac{\lambda}{-\log(1-\delta)}\right) \cdot \log b\right) = O\left(\ell_{\mathsf{acc}} \cdot \left(\frac{\lambda}{-\log(1-\delta)}\right) \cdot \log n\right)$ queries to the random oracle.
- We would also like to minimize the proof size and verifier cost of our Canopy openings, of which there are $\ell_{\mathsf{fold}} = O(\ell_{\mathsf{acc}})$ many, each of which require the verifier to perform $O(\mathsf{qsize}_{\mathsf{fold}} \cdot (1 + \log \mathsf{len}_{\mathsf{cast}} - \log k)) = O(\lambda \cdot (1 + \log n - \log k))$ queries to the random oracle.

In order to simultaneously optimize the foregoing considerations, we choose a Reed–Solomon code $\mathcal{C} = \mathsf{RS}$ with block length $b = \Theta(n/\log n)$ and message length $k = \Theta(n/2^{\sqrt{\log n}})$, rounding both lengths to powers of two and using a suitable smooth evaluation domain. The code has relative distance at least $\delta_{\mathsf{RS}} := 1 - k/b$ and encoding time $T_{\mathsf{RS}} = O(b \log b) = O(n)$. For sufficiently large $n$, we can set

$$\delta := 1 - \left(\frac{3k}{2b}\right)^{1/3} \in (0, \delta_{\mathsf{mca}}(\mathsf{RS})) \ .$$

Since $-\log(1-\delta) = \Theta(\sqrt{\log n})$, this implies that the number of queries that the verifier has to make to the random oracle for the $\ell_{\mathsf{fold}}$ Canopy openings and the interactive accumulation scheme are both balanced at $O(\lambda \cdot \ell_{\mathsf{acc}} \cdot \sqrt{\log n})$.

**Overall metrics.** For fixed accumulation arities, the analysis in Section 8.3.1 gives the following bounds, establishing the metrics in Theorem 1.3:

- Straight-line knowledge soundness error $\kappa_{\mathsf{ACC}} = \mathrm{negl}(\lambda)$ and extraction time $\mathsf{et}_{\mathsf{ACC}} = \mathrm{poly}(\lambda)$.
- Prover cost: $O(\ell_{\mathsf{acc}} \cdot n)$ field operations and $O((\ell_{\mathsf{acc},1} + 1)n + \ell_{\mathsf{acc}} \lambda \sqrt{\log n})$ queries to the random oracle.[14]
- Verifier cost: $O(\ell_{\mathsf{acc}} \cdot \lambda \cdot \log n)$ field operations and $O(\ell_{\mathsf{acc}} \cdot \lambda \cdot \sqrt{\log n})$ queries to the random oracle.
- Decider cost: $O(n)$ field operations and $O(n)$ queries to the random oracle.

[14] The additive $O(n)$ accounts for fresh commitments during folding and interactive accumulation, even when $\ell_{\mathsf{acc},1} = 0$. The remaining term includes verifier replay and Fiat–Shamir queries.

# 9 Implementation and evaluation

We evaluate Canopy by implementing the transformation of Section 7.2 for two IORs. We compare the resulting SNARKs with those obtained by applying the BCS transformation [BCS16] to the corresponding IOPs. Both IOPs follow the structure described in Appendix D.4: the prover sends $\eta$ oracles $\Pi_1, \ldots, \Pi_\eta$, where each $\Pi_i$ has length $n_i$ over an alphabet $\Sigma_i = \mathbb{F}^{2^{t_i}}$ and is a purported codeword of a linear code $\mathcal{C}_0$. The resulting batch of SEP claims is then proven with an IOPP. We consider two choices of codes and IOPPs:

1. *Fast prover, large proofs*: Bolt [GNR26] is a linear-time encodable code with an extremely fast encoding method, but has poor relative distance. Thus, an IOPP for Bolt requires a large number of queries to the oracles, resulting in large proofs.
2. *Slow prover, small proofs*: WHIR [ACFY25] is an IOPP for Reed–Solomon codes in the list-decoding regime, which has a slower prover than Bolt, but requires fewer queries to the oracles since Reed–Solomon codes are maximum-distance separable (MDS), resulting in significantly smaller proofs.

We focus on the commitment and proximity-testing parts of these IOPs, and exclude the application-specific computation that produces their oracles. We additionally evaluate the accumulation construction of Section 8, comparing commitment time, accumulation prover time, and verifier hashes. The implementation is available at `https://github.com/randomshuffle/canopy-system`.

## 9.1 Baselines and experiments

**Reed–Solomon baselines and Canopy modification.** The WHIR IOPP for $\mathsf{SEP}[\mathsf{RS}_0, \delta_0]$ consists of $r = O(\log N)$ rounds of interaction, where $N$ is the message length of the Reed–Solomon code $\mathsf{RS}_0$. The $i$-th round is an IOR from $\mathsf{SEP}[\mathsf{RS}_{i-1}, \delta_{i-1}]$ to $\mathsf{SEP}[\mathsf{RS}_i, \delta_i]$, where $\mathsf{RS}_i$ has a smaller message length than $\mathsf{RS}_{i-1}$ by a factor of $2^{t_i}$ (which we call the folding factor), but the block length is smaller by a factor of 2. This leads to a sequence of Reed–Solomon codes $\mathsf{RS}_0, \mathsf{RS}_1, \ldots, \mathsf{RS}_r$ with decreasing rates. The decreasing rates result in fewer queries to the subsequent oracles, and thus smaller proofs overall.

We instantiate the IOR of Section 7.2 with the first WHIR reduction, run separately on each of the $\eta$ input oracles. It queries each input oracle and outputs one purported codeword of $\mathsf{RS}_1$ per input. We batch the resulting sumcheck claims about $\mathsf{RS}_1$ and prove them with a single run of the remaining WHIR rounds. This serves as an optimized BatchPrx, batching directly at the level of SEP claims instead of using Fig. 9. Thus, the input commitments remain separate, but the commitments to the IOPP oracles are shared. Our two baselines apply the BCS transformation to this IOP with initial rates $1/2$ and $1/4$, respectively, and use Merkle commitments for every oracle.

Our modification instantiates Construction 7.3 with $\eta$ output oracles in place of one. We use $\mathsf{RS}_1$ as the code for Canopy, the same as the code of the oracle sent by the prover in the second round of the IOPP. Thus, the input for BatchPrx then contains $2\eta$ claims about $\mathsf{RS}_1$: $\eta$ from the Canopy openings and $\eta$ from the IOR. All the IOPP oracles are instantiated with Merkle trees.

**Bolt baseline and Canopy modification.** The Bolt-max code is systematic: for an initial interleaving parameter $2^{t_1}$, codewords have a systematic part written as a matrix $X \in \mathbb{F}^{(N/2^{t_1}) \times 2^{t_1}}$, and a parity part $Y = \mathsf{RS}(HX)$, where $H$ is a sparse linear map and Reed–Solomon encoding is applied columnwise [GNR26]. The Bolt IOPP queries both parts and uses code-switching to reduce their consistency and evaluation checks to sumcheck claims about Reed–Solomon codewords, which can then be proven using WHIR (the Bolt paper instead uses the Ligerito [NA25] IOPP for Reed–Solomon codes).

The IOR performs this code-switching on each input, reducing the queries to each of its two parts to a sumcheck claim about a Reed–Solomon codeword. Our baseline first applies one WHIR reduction to each

systematic claim so that the systematic and parity claims use the same code. It then proves all $2\eta$ claims with a single recursive WHIR IOPP, using Merkle commitments for every oracle.

Our modification treats the Bolt IOPP as a black-box and replaces only the input commitments to the systematic and parity parts with two Canopy instances. Every oracle in the Bolt IOPP is then instantiated with a Merkle commitment.

**Accumulation baselines and Canopy modification.** We evaluate a recurring accumulation step with $\eta$ fresh, single-oracle MEP claims and one previous accumulator describing an SEP claim. Our two baselines use Reed–Solomon codes of rates $1/2$ and $1/4$, respectively, and Merkle commitments throughout. Each applies the batching IOR described in Section 8.2 to the $\eta + 1$ full-length claims and compiles it using Merkle trees. The IOR first uses the sumcheck prefix of Appendix D to produce a virtual multilinear evaluation claim about a random linear combination of the inputs. It then uses the same-code WHIR codeswitch to obtain one output SEP claim, whose committed version is the next accumulator.

Let $\mathsf{C} := \mathsf{CR}[\mathsf{SEP}[\mathsf{RS}_0, \delta_0], \mathsf{Canopy}]^{\mathcal{U}}$ and $\mathsf{M} := \mathsf{CR}[\mathsf{SEP}[\mathsf{RS}_C, \delta_C], \mathsf{MT}]^{\mathcal{U}}$, where $\mathsf{RS}_C$ is the code used inside Canopy. Our Canopy-based accumulation scheme, as described in detail in Section 8.3, uses the accumulated relation $\mathsf{C} \times \mathsf{M}$. That is, the accumulator contains one $\mathsf{C}$ claim and one $\mathsf{M}$ claim. The ICAST reduction outputs $\eta$ many $\mathsf{C}$ claims, so our accumulation reduction consists of two IRs: IFOLD and IACC:

$$\mathsf{IFOLD} : \mathsf{C}^{\eta+1} \longrightarrow \mathsf{C} \times \mathsf{M}^{\eta+1} \quad , \quad \mathsf{IACC} : \mathsf{M}^{\eta+2} \longrightarrow \mathsf{M} \quad ,$$

which are defined in detail in Section 8.3. The $\eta + 1$ residual claims returned by IFOLD are combined with the Merkle claim in the input accumulator. Thus, the accumulation step returns one Canopy claim and one Merkle claim, as desired. The input commitments and their encoded witnesses are available to the accumulation prover, and the cost of committing the $\eta$ fresh oracles is measured separately.

## 9.2 Parameter choices

**WHIR and Canopy parameters.** For Reed–Solomon codes, we use the mutual correlated agreement bound of [BCGM26, Theorem 9.2] with $m = 10$. For rate $\rho$, this gives query radius $1 - (1 + 1/(2m))\sqrt{\rho} = 1 - \sqrt{\rho} - \sqrt{\rho}/20$. For proof sizes, we optimize over folding schedules where each $t_i \in \{3, 4\}$, stopping once at most $2^{10}$ message coefficients remain. The Canopy RS codeword has length $N$, as opposed to the $2N$ block length of $\mathsf{RS}_0$. We constrain the Canopy codeword to have the same message length and block length as the second oracle in the WHIR IOPP, so that the two oracles can be batched together efficiently. Thus, the Canopy depth (Construction 6.1) is $d = t_1 + 1$ for an initial fold of $t_1$ variables, and we optimize this depth jointly with the folding schedule. Every selected Canopy configuration chooses $t_1 = 3$ and $d = 4$.

**Bolt-max and Canopy parameters.** For the systematic test, we assume that Bolt-max has relative distance $\delta_{\mathsf{Bolt}} = 0.013$ over our chosen field [GNR26]. We use the general-linear-code proximity bound from Khatam [Zei26, Theorem 1] which improves the mutual correlated agreement radius beyond $\delta_{\mathsf{Bolt}}/3$ to one-and-a-half Johnson bound. Using $\epsilon = \tau = 2^{-32}$, the selected radius is

$$\delta_X = \max\left\{\delta_{\mathsf{Bolt}}/3,\ 1 - (1 - \delta_{\mathsf{Bolt}} + \epsilon)^{1/3} - 3\tau\right\} \simeq 0.00435 \quad .$$

For the parity test we retain the unique-decoding radius $\delta_Y = 1/4$.

We fix $t_1 = 2$ for Bolt-max to obtain reasonable proof sizes for the baseline and our modification. We use Bolt-max's all-one, degree-eight sparse matrix with compression factor eight [GNR26]. Thus, $X$ has $N/4$ rows, $U = HX$ has $N/32$ rows, and its rate-$1/2$ RS encoding $Y$ has $N/16$ rows. Both Canopy Reed–Solomon block lengths are $N/16$, and their message lengths are $N/2^{d_X+2}$ and $N/2^{d_Y+4}$, respectively. We optimize the

two depths independently to minimize the total proof size, including the separate proofs of the Canopy output claims, using folds with $t_i \leq 3$. Every selected systematic depth is $d_X = 3$, and the parity depth $d_Y$ is either 4, 5, 7, or the full depth $\log_2 N - 4$, depending on $(N, \eta)$.

**Accumulation parameters.** We use the same field, hash function, and additive Reed–Solomon encoder as in the SNARK experiments, packing eight field elements into each authenticated leaf. The base code in the Canopy configuration has rate $1/2$ and block length $2N$, and the Canopy code has block length $N$. At depth $d$, its message length is $N/2^{d-1}$ and its rate is $2^{-d+1}$. We choose $d$ to minimize the sum of the expected verifier hashes of IFOLD and IACC, including authentication and Fiat–Shamir. The selected depths are $d = 6$ for $\eta = 1$ and $d = 5$ for $\eta \in \{2, 4, 8\}$, giving Canopy-code rates $1/32$ and $1/16$, respectively. For every code of rate $\rho$, we use the Johnson radius with the same slack as above, $\delta(\rho) = 1 - 1.05\sqrt{\rho}$.

## 9.3 Experimental results

**Organization.** We implement our experiments in rust, building on the `arkworks` field and serialization interfaces and a publicly available implementation of WHIR[15].

**Cryptographic primitives.** All five configurations use $\mathbb{F} = \mathsf{GF}(2^{256})$, with field elements serialized in 32 bytes, and target $\lambda = 128$ bits of round-by-round soundness error. We implement this field as a quadratic extension of $\mathsf{GF}(2^{128})$, with carryless multiplication accelerated by PCLMULQDQ instructions. Reed–Solomon encoding uses additive FFTs on Cantor-subspace domains, including conversion from monomial coefficients to the FFT basis. We use `blake3` 1.5.5 for the 256-bit hashes used in Merkle trees and Canopy. Canopy's H2F function directly interprets a 256-bit digest as a field element of $\mathsf{GF}(2^{256})$. Fiat–Shamir challenges are derived from a BLAKE3 XOF using `spongefish`.

**Machine details.** We ran all reported experiments on an AWS `r7a.32xlarge` instance with AMD EPYC 9R14 processors, 128 physical cores, one hardware thread per core, and 1 TiB of memory. We used 32 Rayon workers pinned to 32 physical cores on one NUMA node, with one benchmark job running at a time.

**SNARK measurement details.** Commitment time includes initial encoding and commitment construction. Since the $\eta$ input oracles arrive in different IOP rounds, their commitments are not computed concurrently. Thus, the commitment time for $\eta$ inputs is $\eta$ times the commitment time for a single input. Opening time includes evaluations, batching, recursive commitments, and query openings. All times include Fiat–Shamir and exclude input generation, setup, and verification. Proof sizes include commitment roots, authentication paths, opened field values, and all non-oracle messages. We count verifier hashes as BLAKE3 hash/XOF finalizations for authentication and Fiat–Shamir.

| | Proof (KiB) | Verifier hashes ($10^3$) | Commit (s) | Open (s) | Total (s) |
|---|---|---|---|---|---|
| WHIR, rate 1/2 + Merkle | 503.68 | 11.02 | 142.87 | 119.32 | 262.19 |
| WHIR, rate 1/4 + Merkle | 338.30 | 7.31 | 225.34 | 199.18 | 424.52 |
| WHIR + Canopy | 410.49 | 7.53 | 184.49 | 144.78 | 329.28 |
| Bolt + Merkle | 12,074.96 | 357.54 | 28.27 | 79.57 | 107.83 |
| Bolt + Canopy | 6,093.64 | 120.55 | 34.45 | 89.06 | 123.51 |

**Table 1:** Comparison at $N = 2^{30}$ and $\eta = 1$.

[15] https://www.github.com/worldfnd/whir/

**Comparison with WHIR.** Figure 6 compares the three WHIR configurations. The absolute timings at $N = 2^{30}$ and $\eta = 1$ appear in Table 1. Batching the Canopy claims with those of the baseline interactive argument reduces proof size relative to the rate-$1/2$ WHIR baseline. This is why we formulate Canopy as a VCR scheme rather than a vector commitment scheme. Without this batching, Canopy's prover time and proof size would increase significantly, eliminating its concrete advantage over the Merkle baselines.

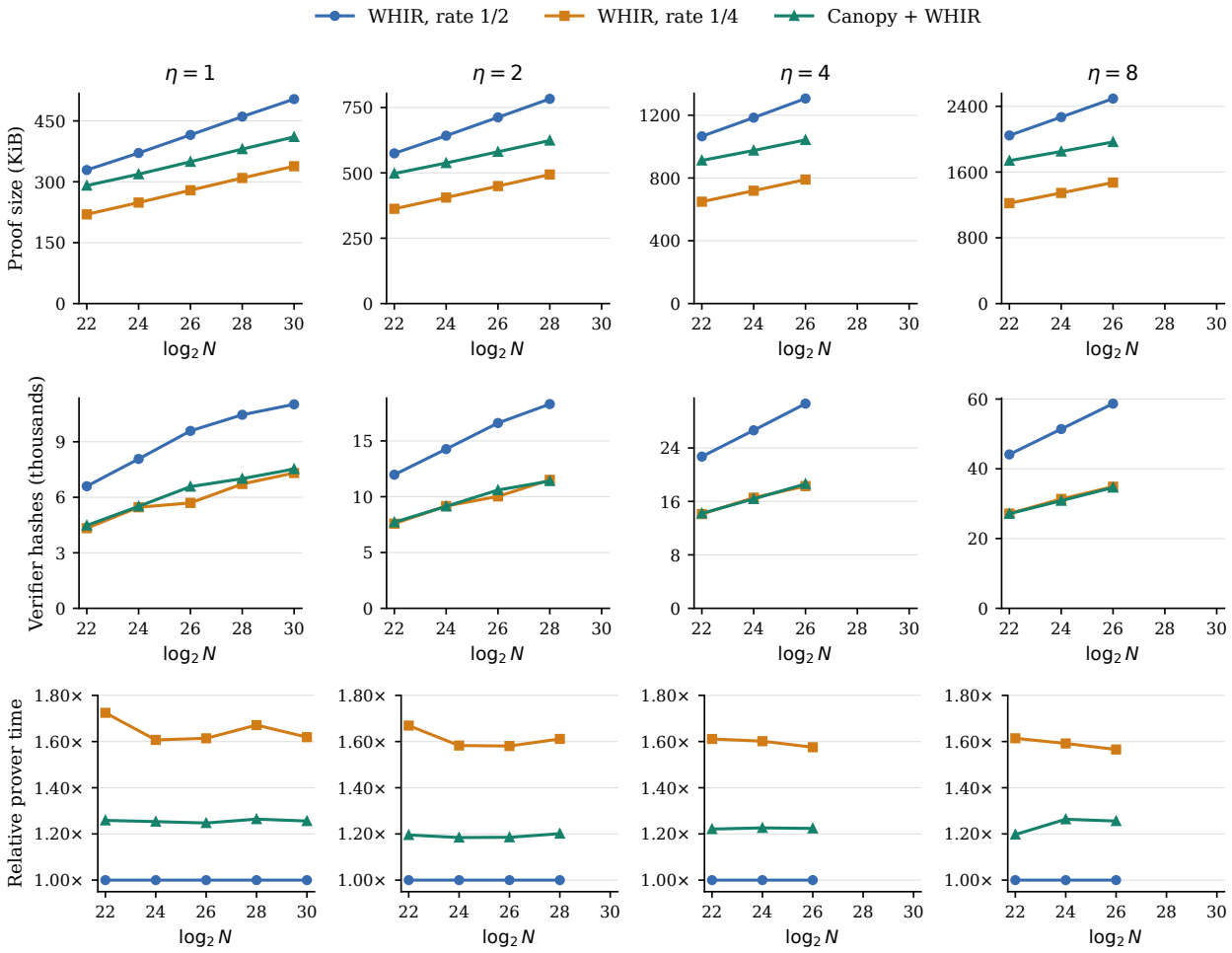


**Figure 6:** WHIR comparison. Prover times are relative to the rate-$1/2$ Merkle baseline at the same $(N, \eta)$.

**Comparison with Bolt-max.** Figure 7 compares the two Bolt configurations. Both exclude the cost of Mulperm, which Bolt uses in its opening proof to verify the sparse-matrix evaluation claim arising from the consistency check between the systematic and parity parts under the sketch matrix $H$. The Bolt evaluation in [GNR26] also does not implement or directly measure Mulperm, and instead estimates its cost from sumcheck benchmarks [GNR26, Remark 8.1]. Since both of our Bolt configurations omit the same component, the comparison between Bolt with Canopy and Bolt with Merkle commitments is apples-to-apples. However, their opening times are not directly comparable with those of any of the three WHIR configurations, which have no such omitted component.

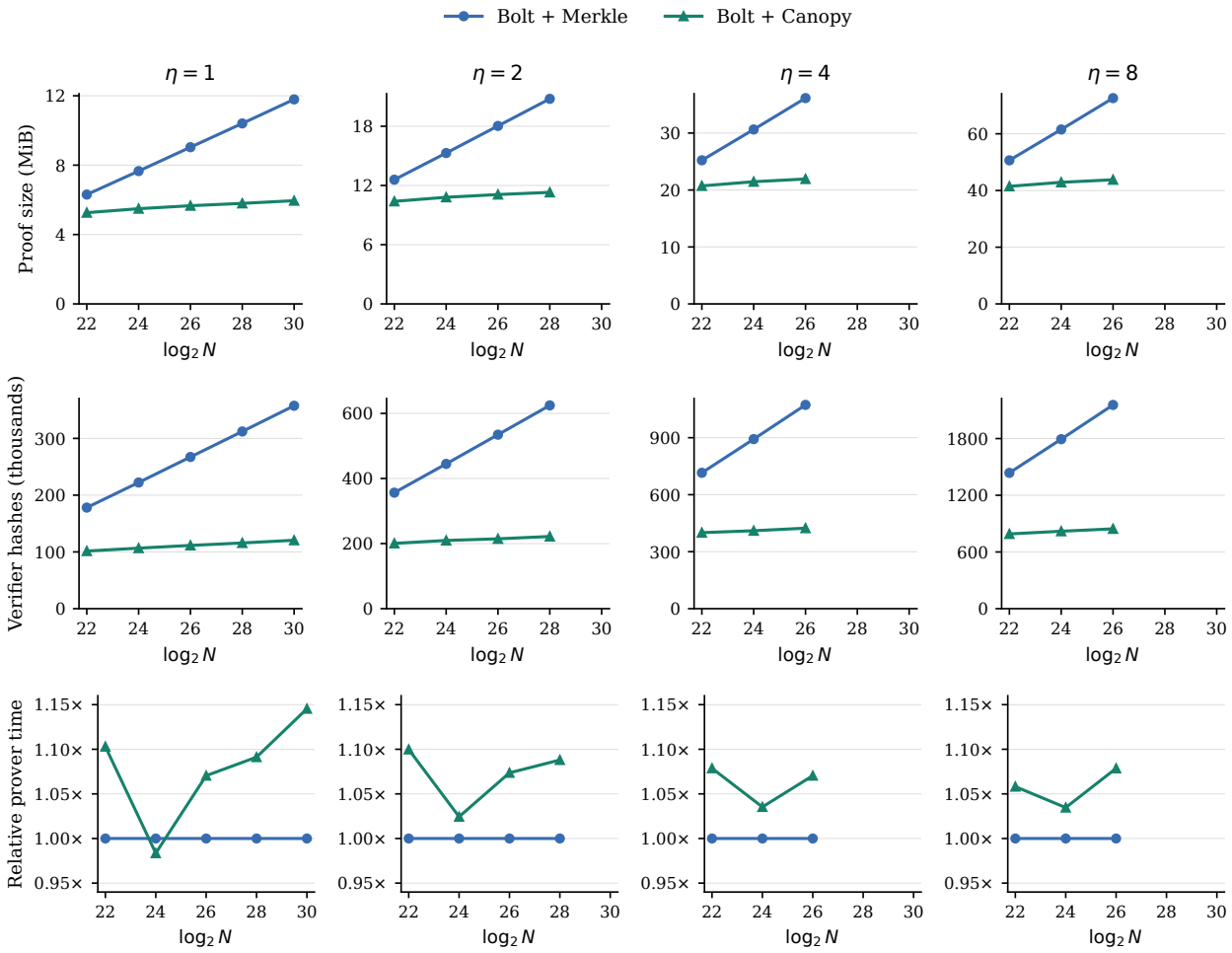


**Figure 7:** Bolt-max comparison. Prover times are relative to the Merkle baseline at the same $(N, \eta)$.

**Accumulation comparisons.** Table 2 and Fig. 8 report the results of the accumulation experiment. Fresh commitment time includes encoding and constructing the commitments to all $\eta$ new oracles. Accumulation prover time starts from those commitments and is the combined time of IFOLD and IACC for Canopy, or the single batching reduction for either Merkle baseline. Both timings include Fiat–Shamir, but preparation of cached prior commitments, input generation, initial claim setup, and the final decider are excluded. Verifier hashes are analytical expectations over the query positions, accounting for repeated queries and shared authentication paths, and include Fiat–Shamir hashes.

| $\eta$ | Configuration | Fresh commit (s) | Accumulate (s) | Expected verifier hashes ($10^3$) |
|---|---|---|---|---|
| 1 | Merkle, rate 1/2 | 1.51 | 1.07 | 8.90 |
| | Merkle, rate 1/4 | 2.82 | 1.36 | 4.72 |
| | Canopy | 2.09 | 1.84 | 5.04 |
| 2 | Merkle, rate 1/2 | 3.07 | 1.30 | 13.34 |
| | Merkle, rate 1/4 | 5.48 | 1.64 | 7.07 |
| | Canopy | 4.22 | 2.19 | 7.03 |
| 4 | Merkle, rate 1/2 | 5.62 | 1.62 | 22.21 |
| | Merkle, rate 1/4 | 9.98 | 1.99 | 11.76 |
| | Canopy | 8.32 | 2.74 | 10.97 |
| 8 | Merkle, rate 1/2 | 11.31 | 3.00 | 39.96 |
| | Merkle, rate 1/4 | 20.16 | 3.85 | 21.14 |
| | Canopy | 15.52 | 4.28 | 18.84 |

**Table 2:** Accumulation at $N = 2^{24}$ with $\eta$ fresh inputs. Fresh commitment and accumulation prover times are measured separately. Canopy accumulation includes both IFOLD and IACC. Verifier hashes are expected counts.

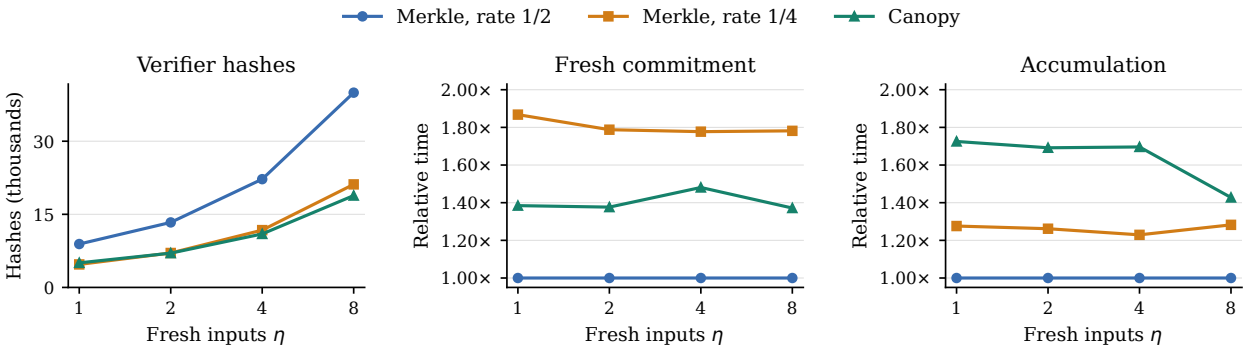


**Figure 8:** Accumulation comparison at $N = 2^{24}$. Commitment and accumulation times are normalized separately to the rate-$1/2$ Merkle baseline at the same $\eta$. Verifier hashes are expected counts.

## AI use disclosure

We used ChatGPT 5.6 Sol, ChatGPT 6 Astra, Claude Opus 5.1, and Claude Fable 5.1 for implementing and benchmarking our construction, drafting proof sketches, revising exposition, formatting, and checking our final proofs for errors. We take full responsibility for the content of the paper, and all the code and proofs were finalized and verified by us. Figures (not including algorithms), plots, and tables were also generated by ChatGPT.

## References


[ACFY24] G. Arnon, A. Chiesa, G. Fenzi, and E. Yogev. "STIR: Reed–Solomon proximity testing with fewer queries". In: CRYPTO '24. URL: https://eprint.iacr.org/2024/390.

[ACFY25] G. Arnon, A. Chiesa, G. Fenzi, and E. Yogev. "WHIR: Reed–Solomon proximity testing with super-fast verification". In: EUROCRYPT '25. URL: https://eprint.iacr.org/2024/1586.

[AHIV17] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: CCS '17. URL: https://eprint.iacr.org/2022/1608.

[BBF19] D. Boneh, B. Bünz, and B. Fisch. "Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains". In: CRYPTO '19. URL: https://eprint.iacr.org/2018/1188.

[BBHR18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. "Fast Reed–Solomon Interactive Oracle Proofs of Proximity". In: ICALP '18. URL: https://eccc.weizmann.ac.il/report/2017/134/.

[BCCT13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. "Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data". In: STOC '13. URL: https://eprint.iacr.org/2012/095.

[BCF+25] M. Brehm, B. Chen, B. Fisch, N. Resch, R. D. Rothblum, and H. Zeilberger. "Blaze: Fast SNARKs from Interleaved RAA Codes". In: EUROCRYPT '25. URL: https://eprint.iacr.org/2024/1609.

[BCFW25] B. Bünz, A. Chiesa, G. Fenzi, and W. Wang. "Linear-Time Accumulation Schemes". In: TCC '25. URL: https://eprint.iacr.org/2025/753.

[BCG+19] E. Ben-Sasson, A. Chiesa, L. Goldberg, T. Gur, M. Riabzev, and N. Spooner. "Linear-Size Constant-Query IOPs for Delegating Computation". In: TCC '19. URL: https://eprint.iacr.org/2019/1230.

[BCG24] E. Boyle, R. Cohen, and A. Goel. "Breaking the $O(\sqrt{n})$-Bit Barrier: Byzantine Agreement with Polylog Bits Per Party". In: *Journal of Cryptology* (2024). URL: https://eprint.iacr.org/2020/130.

[BCGM26] S. Bordage, A. Chiesa, Z. Guan, and I. Manzur. "All Polynomial Generators Preserve Distance with Mutual Correlated Agreement". In: CCC '26. URL: https://eprint.iacr.org/2025/2051.

[BCL+21] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. "Proof-Carrying Data Without Succinct Arguments". In: CRYPTO '21. URL: https://eprint.iacr.org/2020/1618.

[BCMS20] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. "Proof-Carrying Data from Accumulation Schemes". In: TCC '20. URL: https://eprint.iacr.org/2020/499.

[BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner. "Interactive Oracle Proofs". In: TCC '16. URL: https://eprint.iacr.org/2016/116.

[BCTV14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: CRYPTO '14. URL: https://eprint.iacr.org/2014/595.

[BFRW26] B. Bünz, G. Fenzi, R. D. Rothblum, and W. Wang. "TensorSwitch: Nearly Optimal Polynomial Commitments from Tensor Codes". In: CRYPTO '26. URL: https://eprint.iacr.org/2025/2065.

[BFS20] B. Bünz, B. Fisch, and A. Szepieniec. "Transparent SNARKs from DARK Compilers". In: EUROCRYPT '20. URL: https://eprint.iacr.org/2019/1229.

[BMMS26a] A. Baweja, P. Mishra, T. Mopuri, and M. Shtepel. "FICS and FACS: Fast IOPPs and Accumulation via Code-Switching". In: CRYPTO '26. URL: https://eprint.iacr.org/2025/737.

[BMMS26b] A. Baweja, P. Mishra, T. Mopuri, and M. Shtepel. "Query-Optimal IOPPs for Linear-Time Encodable Codes". In: EUROCRYPT '26. URL: https://eprint.iacr.org/2025/1588.

[BMNW25] B. Bünz, P. Mishra, W. Nguyen, and W. Wang. "Arc: Accumulation for Reed–Solomon Codes". In: CRYPTO '25. URL: https://eprint.iacr.org/2024/1731.

[BMRS20] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro. "Coda: Decentralized Cryptocurrency at Scale". Cryptology ePrint Archive, Paper 2020/352. URL: https://eprint.iacr.org/2020/352.

[BR93] M. Bellare and P. Rogaway. "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols". In: CCS '93. URL: https://doi.org/10.1145/168588.168596.

[CBBZ23] B. Chen, B. Bünz, D. Boneh, and Z. Zhang. "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates". In: EUROCRYPT '23. URL: https://eprint.iacr.org/2022/1355.

[CCDW20] W. Chen, A. Chiesa, E. Dauterman, and N. P. Ward. "Reducing Participation Costs via Incremental Verification for Ledger Systems". Cryptology ePrint Archive, Paper 2020/1522. URL: https://eprint.iacr.org/2020/1522.

[CF13] D. Catalano and D. Fiore. "Vector Commitments and Their Applications". In: PKC '13. URL: https://eprint.iacr.org/2011/495.

[CGKY26] A. Chiesa, Z. Guan, C. Knabenhans, and Z. Yu. "On the Fiat–Shamir Security of Succinct Arguments from Functional Commitments". In: CRYPTO '26. URL: https://eprint.iacr.org/2025/902.

[CHM+20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: EUROCRYPT '20. URL: https://eprint.iacr.org/2019/1047.

[CT10] A. Chiesa and E. Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards". In: ICS '10. URL: https://people.csail.mit.edu/tromer/papers/pcd.pdf.

[CY21] A. Chiesa and E. Yogev. "Subquadratic SNARGs in the Random Oracle Model". In: CRYPTO '21. URL: https://eprint.iacr.org/2021/281.

[CY24] A. Chiesa and E. Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024. URL: https://github.com/hash-based-snargs-book.

[DCB25] T. Datta, B. Chen, and D. Boneh. "VerITAS: Verifying Image Transformations at Scale". In: S&P '25. URL: https://eprint.iacr.org/2024/1066.

[DL78] R. A. DeMillo and R. J. Lipton. "A Probabilistic Remark on Algebraic Program Testing". In: *Information Processing Letters* (1978). URL: https://doi.org/10.1016/0020-0190(78)90067-4.

[DMS26] M. Dellepere, P. Mishra, and A. Shirzad. "Garuda and Pari: Faster and Smaller SNARKs via Equifficient Polynomial Commitments". In: USENIX Security '26. URL: https://eprint.iacr.org/2024/1245.

[Dra25] J. Drake. "lean Ethereum". Ethereum Foundation Blog. URL: https://blog.ethereum.org/2025/07/31/lean-ethereum.

[Eag25] L. Eagen. "Glock: Garbled Locks for Bitcoin". Cryptology ePrint Archive, Paper 2025/1485. URL: https://eprint.iacr.org/2025/1485.

[FGZ+26] A. Frolov, J. Guo, X. Zhao, T. Datta, D. Boneh, and I. Miers. "zk-Cinema: Proving Video Provenance in Zero Knowledge". Cryptology ePrint Archive, Report 2026/1598. URL: https://eprint.iacr.org/2026/1598.

[Fs26] M. Frigo and abhi shelat. *Longfellow ZK*. Internet-Draft draft-irtf-cfrg-longfellow-00. Internet Engineering Task Force, Oct. 2026. 72 pp. URL: https://datatracker.ietf.org/doc/draft-irtf-cfrg-longfellow/00/.

[FS86] A. Fiat and A. Shamir. "How to prove yourself: practical solutions to identification and signature problems". In: CRYPTO '86. URL: https://doi.org/10.1007/3-540-47721-7_12.

[GGJ+23] S. Garg, A. Goel, S. Jha, S. Mahloujifar, M. Mahmoody, G.-V. Policharla, and M. Wang. "Experimenting with Zero-Knowledge Proofs of Training". In: CCS '23. URL: https://eprint.iacr.org/2023/1345.

[GGR11] P. Gopalan, V. Guruswami, and P. Raghavendra. "List Decoding Tensor Products and Interleaved Codes". In: *SIAM Journal on Computing* (2011). Preliminary version appeared in STOC '09. URL: https://arxiv.org/abs/0811.4395.

[GKL24] Y. Gao, H. Kan, and Y. Li. "Linear Proximity Gap for Reed–Solomon Codes within the 1.5 Johnson Bound". Cryptology ePrint Archive, Paper 2024/1810. URL: https://eprint.iacr.org/2024/1810.

[GNR26] K. Gurkan, A. Novakovic, and R. D. Rothblum. "Bolt: Faster SNARKs from Sketched Codes". In: CRYPTO '26. URL: https://eprint.iacr.org/2026/310.

[GRS12] V. Guruswami, A. Rudra, and M. Sudan. *Essential coding theory*. Vol. 2. 2012. URL: https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/web-coding-book.pdf.

[GWC19] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. "PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge". Cryptology ePrint Archive, Paper 2019/953. URL: https://eprint.iacr.org/2019/953.

[HSB+26] H. Hafezi, A. Shirzad, B. Bünz, K. Lewi, D. George, and J. Bonneau. "Aegon: Self-Auditable Key Transparency". Cryptology ePrint Archive, Paper 2026/1681. URL: https://eprint.iacr.org/2026/1681.

[HSBB26] H. Hafezi, A. Shirzad, B. Bünz, and J. Bonneau. "IronDict: Transparent Dictionaries from Polynomial Commitments". In: USENIX Security '26. URL: https://eprint.iacr.org/2025/1580.

[KB20] A. Kattis and J. Bonneau. "Proof of Necessary Work: Succinct State Verification with Fairness Guarantees". Cryptology ePrint Archive, Paper 2020/190. URL: https://eprint.iacr.org/2020/190.

[Kil92] J. Kilian. "A note on efficient zero-knowledge proofs and arguments". In: STOC '92. URL: https://doi.org/10.1145/129712.129782.

[KP23] A. Kothapalli and B. Parno. "Algebraic Reductions of Knowledge". In: CRYPTO '23. URL: https://eprint.iacr.org/2022/009.

[KST22] A. Kothapalli, S. T. V. Setty, and I. Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes". In: CRYPTO '22. URL: https://eprint.iacr.org/2021/370.

[Kus18] J. Kuszmaul. "Verkle Trees". Unpublished manuscript. URL: https://math.mit.edu/research/highschool/primes/materials/2018/Kuszmaul.pdf.

[LM19] R. W. F. Lai and G. Malavolta. "Subvector Commitments with Application to Succinct Arguments". In: CRYPTO '19. URL: https://eprint.iacr.org/2018/705.

[LXZ21] T. Liu, X. Xie, and Y. Zhang. "zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy". In: CCS '21. URL: https://eprint.iacr.org/2021/673.

[MBKM19] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updateable Structured Reference Strings". In: CCS '19. URL: https://eprint.iacr.org/2019/099.

[Mic00] S. Micali. "Computationally Sound Proofs". In: *SIAM Journal on Computing* (2000). Preliminary version appeared in FOCS '94. URL: https://doi.org/10.1137/S0097539795284959.

[Mina] O. Labs. "Mina Cryptocurrency". URL: https://minaprotocol.com/.

[MZ25] D. Minzer and K. Z. Zheng. "Improved Round-by-round Soundness IOPs via Reed–Muller Codes". In: FOCS '25. URL: https://eprint.iacr.org/2025/600.

[NA25] A. Novakovic and G. Angeris. "Ligerito: A Small and Concretely Fast Polynomial Commitment Scheme". Cryptology ePrint Archive, Paper 2025/1187. URL: https://eprint.iacr.org/2025/1187.

[NST25] V. Nair, A. Sharma, and B. Thankey. "BrakingBase - a linear prover, poly-logarithmic verifier, field agnostic polynomial commitment scheme". In: ASIACRYPT '25. URL: https://eprint.iacr.org/2024/1825.

[NT16] A. Naveh and E. Tromer. "PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations". In: S&P '16. URL: https://doi.org/10.1109/SP.2016.23.

[RRR16] O. Reingold, R. Rothblum, and G. Rothblum. "Constant-Round Interactive Proofs for Delegating Computation". In: STOC '16. URL: https://eccc.weizmann.ac.il/report/2016/061/.

[Sch80] J. T. Schwartz. "Fast Probabilistic Algorithms for Verification of Polynomial Identities". In: *Journal of the ACM* (1980). URL: https://doi.org/10.1145/322217.322225.

[TFZ+22] N. Tyagi, B. Fisch, A. Zitek-Estrada, J. Bonneau, and S. Tessaro. "VeRSA: Verifiable Registries with Efficient Client Audits from RSA Authenticated Dictionaries". In: CCS '22. URL: https://eprint.iacr.org/2021/627.

[TKPS22] I. Tzialla, A. Kothapalli, B. Parno, and S. Setty. "Transparency Dictionaries with Succinct Proofs of Correct Operation". In: NDSS '22. URL: https://eprint.iacr.org/2021/1263.

[Val08] P. Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: TCC '08. URL: https://doi.org/10.1007/978-3-540-78524-8_1.

[WYX+21] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang. "Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning". In: USENIX Security '21. URL: https://eprint.iacr.org/2021/730.

[YSL+20] M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath. "Coded Merkle Tree: Solving Data Availability Attacks in Blockchains". In: FC '20. URL: https://arxiv.org/abs/1910.01247.

[Zei26] H. Zeilberger. "Khatam: Reducing the Communication Complexity of Code-Based SNARKs". In: CRYPTO '26. URL: https://eprint.iacr.org/2024/1843.

[Zip79] R. Zippel. "Probabilistic algorithms for sparse polynomials". In: EUROSAM '79. URL: https://doi.org/10.1007/3-540-09519-5_73.

[ZYO+25] C. Zhang, X. Yang, D. Oswald, M. Ryan, and P. Jovanovic. "Eva: Efficient Privacy-Preserving Proof of Authenticity for Lossily Encoded Videos". In: S&P '25. URL: https://eprint.iacr.org/2024/1436.

# A Knowledge soundness proof of IOR-to-IR transformation

We first prove knowledge soundness of the generic transformation as stated in Theorem 5.3, and then prove the specialization under the additional assumption of Theorem 5.4.

## A.1 Generic transformation

We prove the round-by-round knowledge soundness of Fig. 2 as stated in Theorem 5.3. We make a different argument for each of the 4 types of verifier challenges in the compiled IR: (a) challenges sent by the VCR commit verifiers, (b) challenges sent by the IOR verifier excluding the final one, (c) the final challenge sent by the IOR verifier, and (d) challenges sent by the VCR reduce verifiers,

**Notation.** Throughout the proof, we will denote the input index and instance of the compiled IR (output by the RBR extraction adversary) by

$$\mathbb{i}_{\mathsf{IR}} \coloneqq (\mathbb{i}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}.\mathbb{i}_{\mathsf{com}}) \ , \ \ \mathbb{x}_{\mathsf{IR}} \coloneqq (\mathbb{x}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}.\mathsf{cm}) \ .$$

We call $(\mathbb{i}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td}))$ a *good* commitment tuple for a VCR scheme $\mathsf{V}_\tau$ if

$$(\mathbb{i}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \mathsf{V}_\tau.\tilde{\mathcal{R}}^h_{\mathsf{com}} \ \ \wedge \ \ (\mathbb{i}, \mathsf{cm}, \mathsf{td}) \notin \mathsf{V}_\tau.\mathcal{R}^h_{\mathsf{bad}} \ . \tag{12}$$

If the index validation for the input commitment fails (Step 1 of the indexer in Fig. 2), every knowledge state of the compiled IR can be set to zero. Thus, we only consider indices passing this check from now on. All extractors return failure on malformed witness encodings. In each round, write $\mathsf{qtr}_0$ and $\mathsf{qtr}_1$ for the adversary's pre- and post-challenge traces, and $\mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}}$ and $\mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}}$ for the indexer and verifier traces supplied in Definition 3.17. Every badness test uses the appropriate interface projection of the full available trace

$$\mathsf{qtr}_{\mathsf{all}} \coloneqq \mathsf{qtr}_0 \,\|\, \mathsf{qtr}_1 \,\|\, \mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}} \,\|\, \mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}} \ .$$

Commitment extraction at an IOR challenge uses only the pre-challenge trace $\mathsf{qtr}_0$.

For the final IOR challenge and the opening phase, we use the following common notation. We fix the order in which the compiled IR executes the opening reductions, and let $\tau_{\mathsf{qry},i} \in \Xi_{\mathsf{qry}}$ be the oracle of the $i$-th opening reduction. Write $\mathsf{V}_{\mathsf{qry},i} \coloneqq \mathsf{V}_{\tau_{\mathsf{qry},i}}$. This order includes every oracle in $\Xi_{\mathsf{qry}}$, even if its query set is empty. Let the corresponding query sets be $Q_i \subseteq [\ell_{\tau_{\mathsf{qry},i}}]$ and their values be $\boldsymbol{a}_i \in \Sigma^{|Q_i|}_{\tau_{\mathsf{qry},i}}$, for each $i \in [|\Xi_{\mathsf{qry}}|]$. Write $\eta \coloneqq (Q_i, \boldsymbol{a}_i)_{i=1}^{|\Xi_{\mathsf{qry}}|}$ for the entire query-answer set for the queried oracles, which is sent before any opening reduction begins. Let $\mathsf{KState}^h_{\tau_{\mathsf{qry},i}}$ be the knowledge state function of the $i$-th opening reduction, and write

$$\mathbb{x}_{\mathsf{qry},i}(\eta) \coloneqq (\mathsf{V}_{\mathsf{qry},i}.\mathsf{cm}, Q_i, \boldsymbol{a}_i)$$

for the input instance for $\mathsf{Query}(\mathsf{V}_{\mathsf{qry},i}.\mathcal{R}_{\mathsf{com}})$ determined by $\eta$. For a transcript prefix $\mathsf{tr}$ containing the entire IOR transcript and a candidate query record $\eta$, let $\mathsf{IORAccept}(\mathsf{tr}, \eta)$ assert that the record passes the checks in the query phase of Fig. 2 (Step 2 of interaction phase) and that the IOR verifier accepts, producing the short output instance $\mathbb{x}'_{\mathsf{out}}$. Also, let $\mathsf{OutAccept}^h(\mathsf{tr}, \eta, \mathsf{w}_{\mathsf{out}})$ assert that

$$\Big((\mathbb{i}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}.\mathbb{i}_{\mathsf{com}}), (\mathbb{x}'_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}.\mathsf{cm}), \mathsf{w}_{\mathsf{out}}\Big) \in \Big(\widetilde{\mathsf{CR}}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^h \cup \widetilde{\mathsf{BCR}}[\vec{\mathsf{V}}_{\mathsf{out}}]^h\Big) \ .$$

**Challenges sent by the VCR commit verifiers.** We first define the knowledge states before the final IOR challenge. Let $S_0$ be the set of input instance-oracle labels, and let $S_j \coloneqq S_0 \cup \{\Pi_i : i \in [j]\}$. For a tuple of

oracle strings $(\boldsymbol{m}_\tau)_{\tau\in S_j}$, write $\vec{\boldsymbol{m}}_{\mathsf{in}}$ for its input-oracle entries and $\mathsf{tr}^{\le j}_{\mathsf{IOR}}(\vec{\boldsymbol{m}})$ for the IOR transcript through its $j$-th challenge, using $M_i = (\boldsymbol{m}_{\Pi_i}, \alpha_i)$ as its $i$-th prover message. The plaintext messages $\alpha_i$ and challenges $\rho_i$ are taken unchanged from the compiled transcript; only the proof-oracle components are replaced by the candidate strings $\boldsymbol{m}_{\Pi_i}$. Let $\mathsf{KState}_{\mathsf{IOR}}$ be the IOR knowledge state function. Immediately after IOR challenge $j < \mathsf{k}$, the witness can take one of two forms:

1. A good knowledge witness has the form

$$\mathsf{w} := \big(\mathsf{good}, \mathsf{w}_{\mathsf{IOR}}, (\boldsymbol{m}_\tau, \mathsf{td}_\tau)_{\tau\in S_j}\big) \quad .$$

   In this case $\mathsf{KState}^h_{\mathsf{IR}} = 1$ if $(\mathbb{i}_\tau, \mathsf{cm}_\tau, (\boldsymbol{m}_\tau, \mathsf{td}_\tau))$ is good for every $\tau \in S_j$ and

$$\mathsf{KState}_{\mathsf{IOR}}(\mathbb{i}_{\mathsf{in}}, \mathbb{x}_{\mathsf{in}}, \vec{\boldsymbol{m}}_{\mathsf{in}}, \mathsf{tr}^{\le j}_{\mathsf{IOR}}(\vec{\boldsymbol{m}}), \mathsf{w}_{\mathsf{IOR}}) = 1 \quad .$$

   We use the same good state until the next IOR prover message is sent and $\Pi_{j+1}$ is committed. In particular, the proof oracle that is currently being committed does not yet have an entry in a good knowledge witness.

2. A bad certificate has the form $\mathsf{w} = (\mathsf{bad}, \tau, \boldsymbol{m}, \mathsf{td})$ for some $\tau \in S_0 \cup \Xi_{\mathsf{pf}}$. For an input oracle or a completed proof-oracle commitment, $\mathsf{KState}^h_{\mathsf{IR}} = 1$ if

$$(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{V}_\tau.\mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \mathsf{V}_\tau.\tilde{\mathcal{R}}^h_{\mathsf{com}} \quad \wedge \quad (\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{V}_\tau.\mathsf{cm}, \mathsf{td}) \in \mathsf{V}_\tau.\mathcal{R}^h_{\mathsf{bad}} \quad . \tag{13}$$

   Note that there is no IOR knowledge state clause in this case. For a proof oracle $\tau$, let $\mathsf{tr}_{\mathsf{com},\tau}$ be its commitment transcript, using the empty transcript for a commitment that has not started and the full transcript for a completed commitment. The state of its bad certificate is

$$\mathsf{V}_\tau.\mathsf{B}^h(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{tr}_{\mathsf{com},\tau}, \boldsymbol{m}, \mathsf{td}) \quad ,$$

   where $\mathsf{V}_\tau.\mathsf{B}^h$ is the round-by-round sparsity predicate of $\mathsf{V}_\tau.\mathsf{Com}$. At a completed commitment, the full-transcript property of $\mathsf{V}_\tau.\mathsf{B}^h$ gives us that

$$(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{tr}_{\mathsf{com},\tau}, \boldsymbol{m}, \mathsf{td}) \text{ satisfies (13)} \iff \mathsf{V}_\tau.\mathsf{B}^h(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{tr}_{\mathsf{com},\tau}, \boldsymbol{m}, \mathsf{td}) = 1 \quad .$$

   At the empty compiled transcript, proof-oracle certificates are false, and the good and input bad-certificate cases are exactly the two cases of the relaxed union in (10). Note that during commitment prover moves, the good state is unchanged and each bad-certificate state satisfies the prover-move property by the prover-move property of $\mathsf{V}_\tau.\mathsf{B}^h$.

Now consider challenge $s$ of the commitment to $\tau = \Pi_{j+1}$, and let $\mathsf{tr}^-_{\mathsf{com},\tau}$ include its $s$-th prover message. The extractor returns the supplied knowledge witness unchanged. The only state that can change at this challenge is the state of a bad certificate for $\tau$. Thus, if an extraction failure occurs, we must have

$$\mathsf{V}_\tau.\mathsf{B}^h(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{tr}^-_{\mathsf{com},\tau} \,\|\, \rho_s, \boldsymbol{m}, \mathsf{td}) = 1 \wedge \mathsf{V}_\tau.\mathsf{B}^h(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{tr}^-_{\mathsf{com},\tau}, \boldsymbol{m}, \mathsf{td}) = 0 \quad .$$

A sparsity adversary simulates the compiled adversary, outputs the commitment prefix $\mathsf{tr}^-_{\mathsf{com},\tau}$, forwards the fresh challenge $\rho_s$, and returns the message and trapdoor in its certificate. The round-$s$ sparsity guarantee bounds this event by $\kappa_{\mathsf{sp},\tau,s}$, and the extraction time is at most $t_*$.

**Challenges sent by the IOR verifier excluding the final one.** Consider IOR challenge $\rho_j$ for $j < \mathsf{k}$. Firstly, if the post-challenge witness contains a bad certificate, the extractor returns it unchanged: every relevant

commitment is already complete before $\rho_j$. Thus, let's consider the good witness case. Before $\rho_j$, the good witness contains commitment witnesses for $S_{j-1}$. Immediately after $\rho_j$, the good witness also contains the commitment witness for $\Pi_j$. Thus, write the input witness of the IR extractor as

$$\mathsf{w}' = \left(\mathsf{good}, \mathsf{w}'_{\mathsf{IOR}}, (\boldsymbol{m}_\tau, \mathsf{td}_\tau)_{\tau \in S_j}\right) \ .$$

We now describe the compiled IR extractor for this step. The extractor recovers the commitment indices by replaying the indexer using $\mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}}$. For each $\tau \in S_j$, it computes

$$(\widehat{\boldsymbol{m}}_\tau, \widehat{\mathsf{td}}_\tau) \leftarrow \mathsf{V}_\tau.\mathsf{Com}.\mathbf{E}(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{V}_\tau.\mathsf{cm}, \mathsf{qtr}_0|_\tau) \ ,$$

where $\mathsf{qtr}_0|_\tau$ denotes the subset of the trace required to invoke the commitment extractor for $\tau$. It tests each extracted trapdoor using $\mathsf{DecBad}_\tau$ with the trace $\mathsf{qtr}_{\mathsf{all}}|_\tau$. If a test returns one, it outputs the corresponding bad certificate. Otherwise, it invokes the round-$j$ IOR extractor on the input oracles $\widehat{\vec{\boldsymbol{m}}}_{\mathsf{in}}$, transcript $\mathsf{tr}^{\leq j}_{\mathsf{IOR}}(\widehat{\vec{\boldsymbol{m}}})$, and witness $\mathsf{w}'_{\mathsf{IOR}}$. If this returns $\mathsf{w}_{\mathsf{IOR}}$, the compiled IR extractor outputs

$$\mathsf{w} \coloneqq \left(\mathsf{good}, \mathsf{w}_{\mathsf{IOR}}, (\widehat{\boldsymbol{m}}_\tau, \widehat{\mathsf{td}}_\tau)_{\tau \in S_{j-1}}\right) \ .$$

We now argue the error and time bounds of the compiled IR extractor. By VCR extractability (see Definition 4.3), the event that the post-challenge state is true but either of the following properties fails for some $\tau \in S_j$ has probability at most $\sum_{\tau \in S_j} \kappa_{\mathsf{com},\tau}$:

1. $(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{V}_\tau.\mathsf{cm}, (\widehat{\boldsymbol{m}}_\tau, \widehat{\mathsf{td}}_\tau)) \in \mathsf{V}_\tau.\tilde{\mathcal{R}}^h_{\mathsf{com}}$, and
2. if $(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{V}_\tau.\mathsf{cm}, \widehat{\mathsf{td}}_\tau) \notin \mathsf{V}_\tau.\mathcal{R}^h_{\mathsf{bad}}$, then $\boldsymbol{m}_\tau \preceq \widehat{\boldsymbol{m}}_\tau$.

Thus, if all extracted trapdoors are good in the sense of (12), we have $\boldsymbol{m}_\tau \preceq \widehat{\boldsymbol{m}}_\tau$ for all $\tau \in S_j$. Consequently, given that

$$\mathsf{KState}_{\mathsf{IOR}}(\mathbb{i}_{\mathsf{in}}, \mathbb{x}_{\mathsf{in}}, \vec{\boldsymbol{m}}_{\mathsf{in}}, \mathsf{tr}^{\leq j}_{\mathsf{IOR}}(\vec{\boldsymbol{m}}), \mathsf{w}'_{\mathsf{IOR}}) = 1 \ ,$$

strong monotonicity of $\mathsf{KState}_{\mathsf{IOR}}$ gives us

$$\mathsf{KState}_{\mathsf{IOR}}(\mathbb{i}_{\mathsf{in}}, \mathbb{x}_{\mathsf{in}}, \widehat{\vec{\boldsymbol{m}}}_{\mathsf{in}}, \mathsf{tr}^{\leq j}_{\mathsf{IOR}}(\widehat{\vec{\boldsymbol{m}}}), \mathsf{w}'_{\mathsf{IOR}}) = 1 \ .$$

The extracted messages $\widehat{\vec{\boldsymbol{m}}}$ are fixed before $\rho_j$, since commitment extraction uses only the pre-challenge trace, fixed commitments, and prescribed indices. By IOR RBRKS and the prover-move property of its knowledge state, the probability of

$$\begin{aligned} &\mathsf{KState}_{\mathsf{IOR}}(\mathbb{i}_{\mathsf{in}}, \mathbb{x}_{\mathsf{in}}, \widehat{\vec{\boldsymbol{m}}}_{\mathsf{in}}, \mathsf{tr}^{\leq j}_{\mathsf{IOR}}(\widehat{\vec{\boldsymbol{m}}}), \mathsf{w}'_{\mathsf{IOR}}) = 1 \\ &\quad \wedge \ \mathsf{KState}_{\mathsf{IOR}}(\mathbb{i}_{\mathsf{in}}, \mathbb{x}_{\mathsf{in}}, \widehat{\vec{\boldsymbol{m}}}_{\mathsf{in}}, \mathsf{tr}^{\leq j-1}_{\mathsf{IOR}}(\widehat{\vec{\boldsymbol{m}}}), \mathsf{w}_{\mathsf{IOR}}) = 0 \end{aligned}$$

is at most $\kappa_{\mathsf{IOR},j}$. Thus, applying a union bound gives error at most

$$\kappa_{\mathsf{IOR},j} + \sum_{\tau \in S_j} \kappa_{\mathsf{com},\tau} \ .$$

The IOR failure probability is bounded without conditioning on successful commitment extraction. Only one commitment extraction per VCR and one IOR extraction are performed, so the time is at most $t_* + \mathsf{et}_{\mathsf{IOR},j} + \sum_{\tau \in S_j} \mathsf{et}_{\mathsf{com},\tau}$.

**The final challenge sent by the IOR verifier.** Immediately before the final IOR challenge, we use the state function defined in the VCR commit verifier case (including both the good-witness and the bad-certificate cases). Immediately after this challenge, the witness is of the form

$$\mathsf{w}' = \left(\mathsf{w}'_{\mathsf{out}}, (\mathsf{w}'_i)_{i=1}^{|\Xi_{\mathsf{qry}}|}\right) \ ,$$

where $\mathsf{w}'_{\mathsf{out}}$ is the witness for the relaxed union $\widetilde{\mathsf{CR}}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}} \cup \widetilde{\mathsf{BCR}}[\vec{\mathsf{V}}_{\mathsf{out}}]^{\mathcal{U}}$ and $\mathsf{w}'_i$ is the witness for $\widetilde{\mathsf{Query}}[\mathsf{V}_{\mathsf{qry},i}.\tilde{\mathcal{R}}_{\mathsf{com}}]^h$. Let $\mathsf{tr}$ be the transcript immediately after this challenge. The query record has not yet been sent, so define

$$\begin{aligned} &\mathsf{KState}^h_{\mathsf{IR}}(\mathbb{i}_{\mathsf{IR}}, \mathbb{x}_{\mathsf{IR}}, \mathsf{tr}, \mathsf{w}') \coloneqq \\ &\quad \exists\, \eta : \quad \mathsf{IORAccept}(\mathsf{tr}, \eta) \wedge \mathsf{OutAccept}^h(\mathsf{tr}, \eta, \mathsf{w}'_{\mathsf{out}}) \\ &\qquad \wedge \bigwedge_{i=1}^{|\Xi_{\mathsf{qry}}|} \mathsf{KState}^h_{\tau_{\mathsf{qry},i}}(\mathsf{V}_{\mathsf{qry},i}.\mathbb{i}_{\mathsf{com}}, \mathbb{x}_{\mathsf{qry},i}(\eta), \varnothing, \mathsf{w}'_i) \ . \end{aligned} \tag{14}$$

By the empty-transcript property of $\mathsf{KState}^h_{\tau_{\mathsf{qry},i}}$, the last clause is equivalent to membership in the corresponding relaxed query relations. Once the prover sends its query record, we require all 3 clauses to hold for that record. This preserves the prover-move property of the knowledge-state function and agrees with the opening state when all local transcripts are empty.

The compiled IR extractor for this step is constructed as follows. Recall that it receives the post-challenge witness $\mathsf{w}'$, the adversary, indexer, and verifier traces, and the transcript $\mathsf{tr}$. If $\mathsf{w}'_{\mathsf{out}}$ is a bad certificate, the compiled IR extractor returns it as the bad certificate for the corresponding input or proof oracle. Also, if $\mathsf{DecBad}$ identifies a bad query witness for any non-index oracle, it returns that certificate instead. These are valid pre-challenge certificates given a true post-challenge state, since all the corresponding commitments are already complete. Otherwise, the compiled IR extractor computes $(\widehat{\boldsymbol{m}}_\tau, \widehat{\mathsf{td}}_\tau)$ for each $\tau \in S_{\mathsf{k}}$ by invoking the VCR extractor with the query trace $\mathsf{qtr}_0|_\tau$, exactly as in the preceding case. If an extracted trapdoor is bad, it returns the corresponding certificate. All badness tests use the appropriate interface projection of $\mathsf{qtr}_{\mathsf{all}}$. If no certificate is returned, the compiled IR extractor parses the IOR output witness $\mathsf{w}'_{\mathsf{IOR}}$ from the good committed output witness, invokes the final IOR extractor on the extracted messages and $\mathsf{w}'_{\mathsf{IOR}}$, and returns

$$\mathsf{w} \coloneqq \left(\mathsf{good}, \mathsf{w}_{\mathsf{IOR}}, (\widehat{\boldsymbol{m}}_\tau, \widehat{\mathsf{td}}_\tau)_{\tau \in S_{\mathsf{k}-1}}\right) \ .$$

To bound the error, suppose that the post-challenge state is true. Every non-index oracle is queried, selected as output, or both. For each $\tau \in S_{\mathsf{k}}$, apply VCR extractability to each supplied $(\boldsymbol{m}_\tau, \mathsf{td}_\tau)$ (use the same $\mathsf{V}_\tau.\mathsf{cm}$ and random-oracle trace for both applications when an oracle is both queried and selected as output). Except with probability

$$2 \sum_{\tau \in S_{\mathsf{k}}} \kappa_{\mathsf{com},\tau} \ ,$$

all extracted witnesses are valid, and each good extracted message extends every supplied message for its oracle (the factor of 2 accounts for two applications of extractability to the same extracted opening when every non-index oracle is both queried and selected as output). A bad extracted witness gives a valid pre-challenge bad-certificate, so it remains to consider the case where all extracted trapdoors are good.

For the index oracle, apply honest commitment binding (item 5 of Theorem 5.3) with the fixed verifier challenges used by the compiled indexer. In its first stage, the binding adversary simulates the compiled

adversary until immediately before the final IOR challenge and computes the full index oracle $\mathbb{I}$, and outputs $\boldsymbol{m}_{\mathsf{hon}} \coloneqq \mathbb{I}$, retaining the compiled adversary's state. The binding experiment then generates the same index commitment as the compiled indexer, since the commitment prover is deterministic given its inputs, oracle, and challenges. In its second stage, the binding adversary samples $\rho_{\mathsf{k}}$, resumes the compiled adversary, and outputs the post-challenge witness of the query relation $(\boldsymbol{m}_{\mathbb{I}}^{\mathsf{qry}}, \mathsf{td}_{\mathbb{I}}^{\mathsf{qry}})$. By honest commitment binding, the probability that this witness is valid and $\boldsymbol{m}_{\mathbb{I}}^{\mathsf{qry}} \not\preceq \mathbb{I}$ is at most $\kappa_{\mathsf{hon},\mathbb{I}}$.

Now, consider any record $\eta$ such that all 3 clauses of (14) hold. The extracted messages agree with all its query answers, using the preceding comparison for the index oracle. Consequently, the simulated IOR verifier produces the same short output instance $\mathbb{x}'_{\mathsf{out}}$ as the true IOR verifier would. Strong monotonicity of $\mathcal{R}_{\mathsf{out}}$ preserves validity of the output witness when the IOR verifier's output oracles are extended to the extracted messages. The full-transcript property of the IOR knowledge state therefore gives

$$\mathsf{KState}_{\mathsf{IOR}}(\mathbb{i}_{\mathsf{in}}, \mathbb{x}_{\mathsf{in}}, \widehat{\vec{\boldsymbol{m}}}_{\mathsf{in}}, \mathsf{tr}_{\mathsf{IOR}}^{\leq \mathsf{k}}(\widehat{\vec{\boldsymbol{m}}}), \mathsf{w}'_{\mathsf{IOR}}) = 1 \ .$$

The extracted messages are fixed before $\rho_{\mathsf{k}}$. By IOR RBRKS and the prover-move property, the probability of

$$\begin{aligned} &\mathsf{KState}_{\mathsf{IOR}}(\mathbb{i}_{\mathsf{in}}, \mathbb{x}_{\mathsf{in}}, \widehat{\vec{\boldsymbol{m}}}_{\mathsf{in}}, \mathsf{tr}_{\mathsf{IOR}}^{\leq \mathsf{k}}(\widehat{\vec{\boldsymbol{m}}}), \mathsf{w}'_{\mathsf{IOR}}) = 1 \\ &\quad \wedge \ \mathsf{KState}_{\mathsf{IOR}}(\mathbb{i}_{\mathsf{in}}, \mathbb{x}_{\mathsf{in}}, \widehat{\vec{\boldsymbol{m}}}_{\mathsf{in}}, \mathsf{tr}_{\mathsf{IOR}}^{\leq \mathsf{k}-1}(\widehat{\vec{\boldsymbol{m}}}), \mathsf{w}_{\mathsf{IOR}}) = 0 \end{aligned}$$

is at most $\kappa_{\mathsf{IOR},\mathsf{k}}$. Thus, applying a union bound gives error at most

$$\kappa_{\mathsf{IOR},\mathsf{k}} + 2 \sum_{\tau \in S_{\mathsf{k}}} \kappa_{\mathsf{com},\tau} + \kappa_{\mathsf{hon},\mathbb{I}} \ .$$

Only one commitment extraction per non-index VCR scheme and one IOR extraction are performed, so the time is at most $t_* + \mathsf{et}_{\mathsf{IOR},\mathsf{k}} + \sum_{\tau \in S_{\mathsf{k}}} \mathsf{et}_{\mathsf{com},\tau}$.

**Challenges sent by the VCR reduce verifier.** Throughout the opening phase, use the fixed witness format

$$\mathsf{w} \coloneqq \left(\mathsf{w}_{\mathsf{out}}, (\mathsf{w}_i)_{i=1}^{|\Xi_{\mathsf{qry}}|}\right) \ .$$

Here $\mathsf{w}_{\mathsf{out}}$ is a witness for the relaxed union of $\mathsf{CR}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]$ and $\mathsf{BCR}[\vec{\mathsf{V}}_{\mathsf{out}}]$, and $\mathsf{w}_i$ is a knowledge state witness for the $i$-th opening reduction $\mathsf{V}_{\mathsf{qry},i}.\mathsf{Rdx}$. Recall that $\mathsf{V}_{\mathsf{qry},i}.\mathsf{Rdx}$ is an IR from $\mathsf{Query}[\mathsf{V}_{\mathsf{qry},i}.\mathcal{R}_{\mathsf{com}}]^{\mathcal{U}}$ to $\mathsf{V}_{\mathsf{qry},i}.\mathcal{R}_{\mathsf{out}}^{\mathcal{U}}$ with RBRKS error $\kappa_{\mathsf{rdx},i,j}$ and extraction time $\mathsf{et}_{\mathsf{rdx},i,j}$ in round $j \in [\mathsf{V}_{\mathsf{qry},i}.\mathsf{k}_{\mathsf{rdx}}]$. For a compiled transcript prefix $\mathsf{tr}$ in this phase, let $\mathsf{tr}_i$ be the local transcript of that reduction. Let $\eta$ be the query record sent in $\mathsf{tr}$. The commitment indices and the index-oracle commitment are those determined by the compiled indexer.

Using the common predicates above, define

$$\begin{aligned} &\mathsf{KState}_{\mathsf{IR}}^{h}(\mathbb{i}_{\mathsf{IR}}, \mathbb{x}_{\mathsf{IR}}, \mathsf{tr}, \mathsf{w}) \coloneqq \\ &\quad \mathsf{IORAccept}(\mathsf{tr}, \eta) \wedge \mathsf{OutAccept}^{h}(\mathsf{tr}, \eta, \mathsf{w}_{\mathsf{out}}) \\ &\quad \wedge \bigwedge_{i=1}^{|\Xi_{\mathsf{qry}}|} \mathsf{KState}_{\tau_{\mathsf{qry},i}}^{h}(\mathsf{V}_{\mathsf{qry},i}.\mathbb{i}_{\mathsf{com}}, \mathbb{x}_{\mathsf{qry},i}(\eta), \mathsf{tr}_i, \mathsf{w}_i) \ . \end{aligned}$$

At the $j$-th challenge of the $t$-th opening reduction for $\mathsf{V}_{\mathsf{qry},t}$, the transcript prefix $\mathsf{tr}$ contains the concatenation of full transcripts of the first $t-1$ opening reductions, and the transcript prefix for $\mathsf{V}_{\mathsf{qry},t}.\mathsf{Rdx}$ before that

challenge. At this stage the knowledge witness $\mathsf{w}$ contains $\mathsf{w}_{\mathsf{out}}$ (defined above), and the concatenation of the witnesses for $\mathsf{V}_{\mathsf{qry},i}.\tilde{\mathcal{R}}_{\mathsf{out}}$ for $i < t$, the RBR knowledge witness for $\mathsf{V}_{\mathsf{qry},t}.\mathsf{Rdx}$ before that challenge, and the witnesses for $\widetilde{\mathsf{Query}}[\mathsf{V}_{\mathsf{qry},i}.\tilde{\mathcal{R}}_{\mathsf{com}}]^h$ for $i > t$. During an opening prover move, only one local transcript changes, so the prover-move property of the state function of $\mathsf{V}_{\mathsf{qry},t}.\mathsf{Rdx}$ implies the prover-move property of the state function the compiled IR.

Now let $\tau \coloneqq \tau_{\mathsf{qry},t}$ and consider the $j$-th verifier challenge of $\mathsf{V}_\tau.\mathsf{Rdx}$. Let $\mathsf{tr}^-_{\tau,j}$ be the transcript prefix for $\mathsf{V}_\tau.\mathsf{Rdx}$ until the $j$-th prover message. The extractor receives as input the following post-challenge witness

$$\mathsf{w}' = (\mathsf{w}'_{\mathsf{out}}, (\mathsf{w}'_i)_{i=1}^{|\Xi_{\mathsf{qry}}|}) \ .$$

It replays the deterministic indexer and verifier computations using their supplied random-oracle traces. Additionally, it parses the index $\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}$ and the instance $\mathbb{x}_{\mathsf{qry},t}(\eta)$ for $\mathsf{Query}[\mathsf{V}_\tau.\mathcal{R}_{\mathsf{com}}]^{\mathcal{U}}$. It invokes the round-$j$ extractor of $\mathsf{V}_\tau.\mathsf{Rdx}$ on $\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}$, $\mathbb{x}_{\mathsf{qry},t}(\eta)$, $\mathsf{tr}^-_{\tau,j} \parallel \rho_j$, $\mathsf{w}'_t$, and the appropriate subsets of the random-oracle traces. If this call returns $\mathsf{w}_t$, it retains $\mathsf{w}_i \coloneqq \mathsf{w}'_i$ for every $i \neq t$ and outputs

$$\mathsf{w} \coloneqq \left(\mathsf{w}'_{\mathsf{out}}, (\mathsf{w}_i)_{i=1}^{|\Xi_{\mathsf{qry}}|}\right) \ .$$

The error bound follows by the following observation: if

$$\mathsf{KState}^h_{\mathsf{IR}}(\mathbb{i}_{\mathsf{IR}}, \mathbb{x}_{\mathsf{IR}}, \mathsf{tr} \parallel \rho_j, \mathsf{w}') = 1 \wedge \mathsf{KState}^h_{\mathsf{IR}}(\mathbb{i}_{\mathsf{IR}}, \mathbb{x}_{\mathsf{IR}}, \mathsf{tr}, \mathsf{w}) = 0 \ ,$$

then based on our construction of the extractor we must have

$$\mathsf{KState}^h_\tau(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathbb{x}_{\mathsf{qry},t}(\eta), \mathsf{tr}^-_{\tau,j} \parallel \rho_j, \mathsf{w}'_t) = 1 \wedge \mathsf{KState}^h_\tau(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathbb{x}_{\mathsf{qry},t}(\eta), \mathsf{tr}^-_{\tau,j}, \mathsf{w}_t) = 0 \ .$$

A component adversary simulates the compiled adversary, outputs the corresponding index and instance for $\mathsf{Query}[\mathsf{V}_\tau.\mathcal{R}_{\mathsf{com}}]^{\mathcal{U}}$ and the pre-challenge transcript, forwards the fresh local challenge, and returns the $t$-th component of the resulting witness. Its additional setup computations are covered by $q_*$. Thus, $\kappa_{\mathsf{rdx},\tau,j}$ is an upper bound on the RBRKS error of the compiled extractor in this case. The extraction time is at most $t_* + \mathsf{et}_{\mathsf{rdx},\tau,j}$. At the full transcript, the full-transcript properties of the opening knowledge states and the definition of $\mathsf{OutAccept}$ give exactly verifier acceptance and membership in the relaxed output relation (11). This completes the proof of Theorem 5.3.

## A.2 Specialized transformation

We prove Theorem 5.4, using the notation and generic construction of Appendix A.1. Fix a round-by-round extraction adversary satisfying the theorem's input-extraction assumption with query bound $q_*$.

**Knowledge states.** Use the generic knowledge states, except that bad certificates are allowed only for proof-oracle labels, and $\mathsf{OutAccept}$ requires membership in $\widetilde{\mathsf{CR}}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^h$. Ordinary knowledge witnesses before the final IOR challenge have the form $(\mathsf{w}_{\mathsf{IOR}}, (\boldsymbol{m}_\tau)_{\tau \in S_j}, (\mathsf{td}_\tau)_{\tau \in S_j})$. Proof-oracle certificates retain the form $(\mathsf{bad}, \tau, \boldsymbol{m}, \mathsf{td})$ and are governed by the same sparsity predicates as before. At the empty transcript all these certificate states are zero, and the ordinary state is exactly membership in $\widetilde{\mathsf{CR}}[\mathcal{R}_{\mathsf{in}}, \vec{\mathsf{V}}_{\mathsf{in}}]^h$. After the final IOR challenge, the committed output witness is untagged and must belong to the ordinary committed output relation. The prover-move properties are preserved, and at the full transcript the state is exactly verifier acceptance together with membership in

$$\widetilde{\mathsf{CR}}[\mathcal{R}_{\mathsf{out}}, \vec{\mathsf{V}}_{\mathsf{out}}]^h \times \vec{\mathsf{V}}_{\mathsf{qry}}.\tilde{\mathcal{R}}^h_{\mathsf{out}} \ .$$

**Extracting input commitments.** At IOR challenge $j$, for each $\tau \in S_0$, compute

$$(\widehat{\boldsymbol{m}}_\tau, \widehat{\mathsf{td}}_\tau) \leftarrow \mathsf{V}_\tau.\mathsf{Com}.\mathbf{E}(\mathsf{V}_\tau.\mathbb{i}_{\mathsf{com}}, \mathsf{V}_\tau.\mathsf{cm}, \mathsf{qtr}_0|_\tau) \ .$$

Here $\mathsf{qtr}_0$ includes the entire prior query history, including input generation and all subsequent computation before the fresh IOR challenge. For any one opening $(\boldsymbol{m}_\tau, \mathsf{td}_\tau)$ supplied in the post-challenge witness, VCR extractability bounds the event that the supplied opening is valid and the extracted opening is either invalid or non-bad but fails to extend $\boldsymbol{m}_\tau$, by $\kappa_{\mathsf{com},\tau}$. The additional hypothesis bounds the event that the extracted opening is valid and bad by $\beta_\tau$. The union of these events contains every outcome in which the supplied opening is valid and the extracted opening either is not good in the sense of (12) or does not extend the supplied message. Thus, the input-extraction failure bound is $\kappa_{\mathsf{com},\tau} + \beta_\tau$.

**IOR challenges before the final one.** If the supplied witness is a proof-oracle certificate, return it unchanged. Otherwise, run the generic extractor on all commitments in $S_j$, testing extracted trapdoors with $\mathsf{DecBad}_\tau$ and the full available trace. A bad extracted proof-oracle opening is returned as a certificate, while a bad extracted input opening causes the extractor to return failure. On a true post-challenge state, the preceding input-extraction bound accounts for the latter case, as well as invalid extraction and failure of extension for input labels. For proof labels, the generic commitment-extraction argument applies unchanged. Outside these failure events, either a valid proof-oracle certificate is returned or all extracted openings are good and extend their supplied openings. In the latter case, the same strong-monotonicity and IOR RBRKS argument gives an ordinary pre-challenge witness. Consequently, the round-$j$ error is at most

$$\kappa_{\mathsf{IOR},j} + \sum_{\tau \in S_j} \kappa_{\mathsf{com},\tau} + \sum_{\tau \in S_0} \beta_\tau \ .$$

**The final IOR challenge.** The supplied committed output witness is ordinary, so it cannot itself be a bad certificate. The extractor may return a bad supplied query opening only when it concerns a proof oracle. It does not return bad supplied input openings as certificates; instead, it extracts their commitments from the pre-challenge trace and uses the input-extraction bound above. It handles bad extracted openings as in the preceding case and retains the generic treatment of the honestly committed index oracle. Every non-index label has at least one supplied opening, since $\Xi_{\mathsf{all}} = \Xi_{\mathsf{qry}} \cup \Xi_{\mathsf{out}}$, and at most two, since the output selection has no repetitions. Apply the input-extraction bound to each supplied opening of an input label and the generic extraction bound to each supplied opening of a proof label. Together with honest index binding and the final IOR RBRKS guarantee, this gives error at most

$$\kappa_{\mathsf{IOR},\mathsf{k}} + 2 \sum_{\tau \in S_\mathsf{k}} \kappa_{\mathsf{com},\tau} + 2 \sum_{\tau \in S_0} \beta_\tau + \kappa_{\mathsf{hon},\mathbb{I}} \ .$$

The IOR failure probabilities are bounded without conditioning on successful commitment extraction, exactly as in the generic proof.

**Commitment and opening challenges.** The commitment-round extractor is still the identity: only the certificate state for the proof commitment currently in progress can change at its challenge. Its error remains $\kappa_{\mathsf{sp},\tau,s}$. During an opening reduction, the ordinary committed output witness is carried unchanged and only the current opening witness is extracted, so the error remains $\kappa_{\mathsf{rdx},\tau,s}$. There are no additional component extractor calls, and all other computations are covered by $t_*$. This proves the claimed replacement of $\kappa_{\mathsf{com},\tau}$ by $\kappa_{\mathsf{com},\tau} + \beta_\tau$ for input labels, with the extraction times of Theorem 5.3, and completes the proof of Theorem 5.4.

# B Proofs for Canopy

We use the parameters and assumptions of Section 6. The extractors below are deterministic, and neither they nor the Merkle opening algorithms make random-oracle queries. Thus, the extraction and sparsity reductions preserve the adversary's query budget. Algorithms return fixed defaults on malformed inputs or undefined outputs.

**Commitment completeness.**

*Proof of Lemma 6.2.* An honest execution satisfies $\mathcal{R}^h_{\mathsf{com}}$ by construction, for every $h$. Its top word $\boldsymbol{c} = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y})$ has distance at least $\delta_{\mathcal{C}} \geq \delta$ from every other codeword, so no other message witnesses membership in the bad relation. □

**Opening completeness.**

*Proof of Lemma 6.5.* Let $(\mathbb{i}_{\mathsf{com}}, (\mathsf{cm}, Q, \boldsymbol{a}), (\boldsymbol{m}, \mathsf{td})) \in \mathsf{Query}[\mathcal{R}_{\mathsf{com}}]^h$ and consider an honest execution of Fig. 4. Merkle completeness and $\boldsymbol{a}[v_i \,\|\, S_{v_i}] = \boldsymbol{m}_{v_i}[S_{v_i}]$ give $b = 1$ for every $h$. By the identity $p(\boldsymbol{z}) = \sum_{\boldsymbol{i}} p(\boldsymbol{i}) \cdot \mathrm{eq}(\boldsymbol{i}, \boldsymbol{z})$ of Section 3 and the conditions of $\mathcal{R}^h_{\mathsf{com}}$,

$$\sum_{\boldsymbol{b} \in \{0,1\}^{m-d}} f_\gamma(\boldsymbol{b}, \hat{y}(\boldsymbol{b})) = \hat{y}(\boldsymbol{z}_{\mathsf{ood}}) + \sum_{i \in [t]} \gamma^i \cdot \hat{y}(v_i) = \zeta_{\mathsf{ood}} + \sum_{i \in [t]} \gamma^i \cdot \mathsf{H2F}(\mathsf{rt}_{v_i}) = \zeta_\gamma \ .$$

Since $\boldsymbol{c} = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y})$, the output satisfies the strict $\mathsf{SEP}[\mathcal{C}, \delta]$ relation. The honest top commitment, evaluated under its original random-oracle interface, and its empty bad relation then give membership in $\mathcal{R}^h_{\mathsf{out}}$ by Definition 5.1. □

**Commitment extractability.**

*Proof of Lemma 6.3.* The extractor list decodes the extracted top word and filters the list using the out-of-domain evaluation, following STIR [ACFY24]. It then uses a surviving message to recover the block roots. Multiple survivors place its trapdoor in the bad relation, where Definition 4.3 requires validity but no extension guarantee, so the error bound includes only Merkle extraction errors.

**The extractor.** On input $(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, \mathsf{qtr})$ with $\mathsf{cm} = (\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}})$, the extractor $\mathsf{Com}.\mathbf{E}$ proceeds in four steps.

1. Extract the top word, $(\boldsymbol{c}', \mathsf{td}'_{\mathsf{top}}) \leftarrow \mathsf{MT}_{\mathsf{top}}.\mathsf{Extract}(\mathsf{rt}, \mathsf{qtr})$.
2. Compute $\Lambda(\mathcal{C}, \boldsymbol{c}', \delta)$ with the list-decoding algorithm, and let $Y$ be the set of messages $\boldsymbol{y}'$ with $\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y}') \in \Lambda(\mathcal{C}, \boldsymbol{c}', \delta)$ and $\hat{y}'(\boldsymbol{z}_{\mathsf{ood}}) = \zeta_{\mathsf{ood}}$. Let $\boldsymbol{y}'$ be the first element of $Y$ in a fixed ordering.
3. For every $v \in \{0,1\}^{m-d}$ whose value $\boldsymbol{y}'[v]$ is in the image of $\mathsf{H2F}$, set $\mathsf{rt}'_v := \mathsf{H2F}^{-1}(\boldsymbol{y}'[v])$ and extract the block: $(\boldsymbol{m}'_v, \mathsf{td}'_v) \leftarrow \mathsf{MT}_{\mathsf{btm}}.\mathsf{Extract}(\mathsf{rt}'_v, \mathsf{qtr})$. For every other $v$, set $(\mathsf{rt}'_v, \boldsymbol{m}'_v, \mathsf{td}'_v) := (0^\sigma, \perp^{2^d}, \perp)$.
4. Output the message $\boldsymbol{m}'$ with $\boldsymbol{m}'[v \,\|\, s] := \boldsymbol{m}'_v[s]$ and the trapdoor $\mathsf{td}' := (\{(\mathsf{rt}'_v, \mathsf{td}'_v)\}_v, \boldsymbol{y}', \boldsymbol{c}', \mathsf{td}'_{\mathsf{top}})$.

When $Y$ is nonempty, the extracted witness lies in $\tilde{\mathcal{R}}^h_{\mathsf{com}}$ by Lemma 4.12(i) and the choice of $\boldsymbol{y}'$; empty blocks impose no conditions on their roots or trapdoors. Moreover, $\mathsf{td}'$ lies in $\mathcal{R}^h_{\mathsf{bad}}$ exactly when $|Y| \geq 2$, since the other elements of $Y$ are precisely the messages witnessing badness.

**The bad events.** Consider the experiment of Definition 4.3 with a stateful $q$-query adversary $\mathcal{A}$, whose first stage outputs $\mathsf{cm}$ with trace $\mathsf{qtr}$ and whose second stage outputs $(\boldsymbol{m}, \mathsf{td})$. Call the adversary's witness *valid* if $(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \tilde{\mathcal{R}}^h_{\mathsf{com}}$, and define two events.

- $E_{\mathsf{top}}$ is the event that the adversary's witness is valid and $\boldsymbol{c} \not\approx \boldsymbol{c}'$.

- $E_{\mathsf{btm}}$ is the event that the adversary's witness is valid, $Y$ is nonempty, and some $v$ has $\boldsymbol{y}'[v]$ in the image of $\mathsf{H2F}$, $\mathsf{rt}_v = \mathsf{rt}'_v$, and $\boldsymbol{m}_v \not\preceq \boldsymbol{m}'_v$.

Suppose that the adversary's witness is valid and neither event occurs. Then $\boldsymbol{c} \preceq \boldsymbol{c}'$, which gives $\Lambda(\mathcal{C}, \boldsymbol{c}, \delta) \subseteq \Lambda(\mathcal{C}, \boldsymbol{c}', \delta)$. Validity of the adversary's witness therefore places $\boldsymbol{y}$ in $Y$, so the extracted witness is valid. If its trapdoor is bad, we are done; otherwise $Y = \{\boldsymbol{y}\}$ and $\boldsymbol{y}' = \boldsymbol{y}$. For every $v$ with $\mathrm{Dom}(\boldsymbol{m}_v) \neq \varnothing$, the relaxed relation gives $\boldsymbol{y}'[v] = \mathsf{H2F}(\mathsf{rt}_v)$. Step 3 therefore extracts from $\mathsf{rt}'_v = \mathsf{rt}_v$, and the absence of $E_{\mathsf{btm}}$ gives $\boldsymbol{m}_v \preceq \boldsymbol{m}'_v$. Hence $\boldsymbol{m} \preceq \boldsymbol{m}'$, and extraction failure is contained in $E_{\mathsf{top}} \cup E_{\mathsf{btm}}$.

**Bounding the bad events.** Applying Lemma 4.12(ii) to the top commitment and the adversary's eventual opening $(\boldsymbol{c}, \mathsf{td}_{\mathsf{top}})$ gives $\Pr[E_{\mathsf{top}}] \leq \kappa_{\mathsf{MT}}(\sigma, b, q)$. For $E_{\mathsf{btm}}$, apply Lemma 4.12(ii) with $n = 2^{m-d}$ to the following adversary. Its commitment phase runs the first stage of $\mathcal{A}$ and the first three steps of $\mathsf{Com}.\mathbf{E}$, and then outputs the roots $\mathsf{rt}'_v$ of all blocks in sequence without queries in between. For each $v$ with $\boldsymbol{y}'[v]$ in the image of $\mathsf{H2F}$, the Merkle extractor therefore returns the same $\boldsymbol{m}'_v$ as in step 3. Its opening phase runs the second stage of $\mathcal{A}$ and outputs $(v, \boldsymbol{m}_v, \mathsf{td}_v)$ for the first $v$ satisfying the block conditions in $E_{\mathsf{btm}}$. This block is nonempty and its opening is valid, so $\Pr[E_{\mathsf{btm}}] \leq \kappa_{\mathsf{MT}}(\sigma, 2^d, q, 2^{m-d})$. Summing the two bounds proves the claim. □

**Sparsity of the bad relation.**

*Proof of Lemma 6.4.* We follow the out-of-domain sampling argument of STIR [ACFY24, Lemma 4.5], using the decoding list of the extracted top word and comparing multilinear message extensions.

**The predicate.** Set $\mathsf{B}^h(\mathbb{i}_{\mathsf{com}}, \mathsf{tr}, \boldsymbol{m}, \mathsf{td}) = 0$ on malformed inputs, on the empty transcript, and on $(\mathsf{rt})$. After $\boldsymbol{z}_{\mathsf{ood}}$ and until the final empty challenge, it is 1 exactly when $(\mathsf{rt}, \mathsf{td}_{\mathsf{top}})$ opens $\boldsymbol{c}$ under $\mathsf{MT}_{\mathsf{top}}$, $\boldsymbol{c} \sim_\delta \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y})$, and the bad-relation condition holds. This condition is well defined before $\zeta_{\mathsf{ood}}$ is sent because badness does not depend on that value. On the full transcript, the predicate additionally requires $(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}, \mathsf{td})) \in \tilde{\mathcal{R}}^h_{\mathsf{com}}$. These choices satisfy Definition 4.4: neither prover moves nor the final empty challenge can turn the predicate from 0 to 1, giving $\kappa_{\mathsf{sp},2} = 0$.

**The first round.** Consider a stateful $q$-query adversary $\mathcal{A}$ that outputs $(\mathsf{rt})$ with trace $\mathsf{qtr}_0$, receives $\boldsymbol{z}_{\mathsf{ood}} \xleftarrow{\$} \mathbb{F}^{m-d}$, and outputs $(\boldsymbol{m}, \mathsf{td})$. Let $(\boldsymbol{c}^\circ, \cdot) \leftarrow \mathsf{MT}_{\mathsf{top}}.\mathsf{Extract}(\mathsf{rt}, \mathsf{qtr}_0)$, and let $F$ be the event that the predicate is 1 and $\boldsymbol{c} \not\preceq \boldsymbol{c}^\circ$. Applying Lemma 4.12(ii) to the top commitment and eventual opening $(\boldsymbol{c}, \mathsf{td}_{\mathsf{top}})$, with the reduction sampling $\boldsymbol{z}_{\mathsf{ood}}$ itself, gives $\Pr[F] \leq \kappa_{\mathsf{MT}}(\sigma, b, q)$. If the predicate is 1 and $F$ does not occur, then $\boldsymbol{c} \preceq \boldsymbol{c}^\circ$, so the encodings of $\boldsymbol{y}$ and of the message $\boldsymbol{y}'$ witnessing the bad relation lie in $\Lambda(\mathcal{C}, \boldsymbol{c}^\circ, \delta)$. This set has at most $|\Lambda(\mathcal{C}, \delta)|$ codewords and is determined by $\mathsf{rt}$ and $\mathsf{qtr}_0$, hence fixed before $\boldsymbol{z}_{\mathsf{ood}}$ is sampled. For any two distinct messages with encodings in this set, the difference of their multilinear extensions is a nonzero multilinear polynomial, which vanishes at $\boldsymbol{z}_{\mathsf{ood}}$ with probability at most $(m-d)/|\mathbb{F}|$ by Lemma 3.4. A union bound over the at most $\binom{|\Lambda(\mathcal{C}, \delta)|}{2}$ pairs, added to $\Pr[F]$, gives the claimed $\kappa_{\mathsf{sp},1}$. □

**Deciding the bad relation.**

*Proof of Lemma 6.7.* The algorithm $\mathsf{DecBad}$ rejects inputs with $\mathbb{i} \neq \mathbb{i}_{\mathsf{com}}$ or malformed commitments or trapdoors. Otherwise, it computes $\Lambda(\mathcal{C}, \boldsymbol{c}, \delta)$ and accepts exactly when it contains $\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y}'')$ for some $\boldsymbol{y}'' \neq \boldsymbol{y}$ with $\hat{y}''(\boldsymbol{z}_{\mathsf{ood}}) = \hat{y}(\boldsymbol{z}_{\mathsf{ood}})$. This is the defining condition of badness and requires neither $\mathsf{qtr}$ nor random-oracle queries. The stated running time follows from list decoding and multilinear evaluation, giving assumption 4 of Theorem 5.3 when $T_{\mathsf{LD}}$ is polynomial. □

**Knowledge soundness of the opening reduction.**

*Proof of Lemma 6.6.* The output witness already supplies the canopy message and top opening, so the extractor only recovers the touched blocks. To show that this message matches the claimed evaluations, we view the sumcheck condition as a polynomial in $\gamma$ and bound the probability of an accidental root using the decoding list fixed before the challenge. List decoding is needed only in the analysis. We write $\mathbb{x} = (\mathsf{cm}, Q, \boldsymbol{a})$ with $\mathsf{cm} = (\mathsf{rt}, \boldsymbol{z}_{\mathsf{ood}}, \zeta_{\mathsf{ood}})$ for the instance and $\alpha_1 = \{(\mathsf{rt}_{v_i}, \mathsf{pf}_i)\}_{i\in[t]}$ for the single prover message, with $v_1 < \cdots < v_t$ and $(S_{v_i})_{i\in[t]}$ as in Construction 6.1.

**Knowledge state function.** On the empty transcript, $\mathsf{KState}^h(\mathbb{i}, \mathbb{x}, \mathsf{tr}, \mathsf{w})$ is 1 if and only if $(\mathbb{i}, \mathbb{x}, \mathsf{w}) \in \widetilde{\mathsf{Query}}[\tilde{\mathcal{R}}_{\mathsf{com}}]^h$, and on $(\alpha_1)$ it takes the same value. On a full transcript, it is 1 if and only if $\mathsf{Rdx}.\mathbf{V}$ accepts and $(\mathbb{i}_{\mathsf{out}}, \mathbb{x}_{\mathsf{out}}, \mathsf{w})$ lies in the relaxed relation of $\mathcal{R}^h_{\mathsf{out}}$. This satisfies Definition 3.16, since the prover message leaves the state unchanged.

**Extractor.** The extractor $\mathcal{E}_{\text{rbr}}$ uses only the adversary's trace $\mathsf{qtr} := \mathsf{qtr}_0 \,\|\, \mathsf{qtr}_1$ and parses the output witness as $(\boldsymbol{y}^*, \boldsymbol{c}^*, \mathsf{td}^*_{\mathsf{top}})$. For each touched block $v_i$, it sets $\mathsf{rt}^*_{v_i} := \mathsf{rt}_{v_i}$ and computes $(\boldsymbol{m}^*_{v_i}, \mathsf{td}^*_{v_i}) \leftarrow \mathsf{MT}_{\mathsf{btm}}.\mathsf{Extract}(\mathsf{rt}_{v_i}, \mathsf{qtr})$. For every untouched block $v \notin V(Q)$, it sets $(\mathsf{rt}^*_v, \boldsymbol{m}^*_v, \mathsf{td}^*_v) := (0^\sigma, \perp^{2^d}, \perp)$. It assembles these blocks and $(\boldsymbol{y}^*, \boldsymbol{c}^*, \mathsf{td}^*_{\mathsf{top}})$ into the witness $\mathsf{w} := (\boldsymbol{m}^*, \mathsf{td}^*)$ as in Construction 6.1.

**Successful runs.** Consider the experiment of Definition 3.17 with a stateful $q$-query adversary $\mathcal{A}$, whose first stage outputs $(\mathbb{i}, \mathbb{x}, (\alpha_1))$ with trace $\mathsf{qtr}_0$ and whose second stage receives $\gamma$ and outputs $\mathsf{w}'$ with trace $\mathsf{qtr}_1$. Call the run *successful* if $|(\mathbb{i}, \mathbb{x})| \le n$ and the knowledge state on the full transcript with $\mathsf{w}'$ is 1. In a successful run, all block openings verify, $(\mathsf{rt}, \mathsf{td}^*_{\mathsf{top}})$ opens $\boldsymbol{c}^*$ under $\mathsf{MT}_{\mathsf{top}}$, and $\boldsymbol{c}^* \sim_\delta \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y}^*)$. For a message $\boldsymbol{y}' \in \mathbb{F}^{2^{m-d}}$, define the univariate polynomial

$$p_{\boldsymbol{y}'}(\Gamma) := \big(\hat{y}'(\boldsymbol{z}_{\mathsf{ood}}) - \zeta_{\mathsf{ood}}\big) + \sum_{i\in[t]} \Gamma^i \cdot \big(\hat{y}'(v_i) - \mathsf{H2F}(\mathsf{rt}_{v_i})\big) \quad .$$

For each fixed $\boldsymbol{y}'$, this polynomial has degree at most $t$ and is determined by $\mathbb{x}$ and $\alpha_1$. Its coefficients vanish exactly when $\hat{y}'(\boldsymbol{z}_{\mathsf{ood}}) = \zeta_{\mathsf{ood}}$ and $\boldsymbol{y}'[v_i] = \mathsf{H2F}(\mathsf{rt}_{v_i})$ for every $i \in [t]$. In a successful run, the output sumcheck condition gives $p_{\boldsymbol{y}^*}(\gamma) = 0$ by multilinearity, as in the completeness proof.

**Bad events.** Let $(\boldsymbol{c}^\circ, \cdot) \leftarrow \mathsf{MT}_{\mathsf{top}}.\mathsf{Extract}(\mathsf{rt}, \mathsf{qtr}_0)$ be the top word extracted from the trace before the challenge. We show that a successful run in which extraction fails lies in the union of the following three events.

- $B_1$: the run is successful and $\boldsymbol{c}^* \not\preceq \boldsymbol{c}^\circ$.
- $B_2$: $|(\mathbb{i}, \mathbb{x})| \le n$, the instance and $\alpha_1$ parse, and some $\boldsymbol{y}'$ with $\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y}') \in \Lambda(\mathcal{C}, \boldsymbol{c}^\circ, \delta)$ has $p_{\boldsymbol{y}'}$ nonzero and $p_{\boldsymbol{y}'}(\gamma) = 0$.
- $B_3$: the run is successful and $\boldsymbol{m}^*_{v_i}[S_{v_i}] \neq \boldsymbol{a}[v_i \,\|\, S_{v_i}]$ for some $i \in [t]$.

Consider a successful run outside $B_1 \cup B_2 \cup B_3$. Since $\boldsymbol{c}^* \preceq \boldsymbol{c}^\circ$, the encoding of $\boldsymbol{y}^*$ lies in $\Lambda(\mathcal{C}, \boldsymbol{c}^\circ, \delta)$, so $p_{\boldsymbol{y}^*}$ is the zero polynomial. Hence $\hat{y}^*(\boldsymbol{z}_{\mathsf{ood}}) = \zeta_{\mathsf{ood}}$ and $\boldsymbol{y}^*[v_i] = \mathsf{H2F}(\mathsf{rt}_{v_i})$ for every $i \in [t]$. Every block of $\boldsymbol{m}^*$ with a defined coordinate is a touched block $v_i$, and $(\mathsf{rt}_{v_i}, \mathsf{td}^*_{v_i})$ opens $\boldsymbol{m}^*_{v_i}$ under $\mathsf{MT}_{\mathsf{btm}}$ by Lemma 4.12(i). With the conditions inherited from $\mathsf{w}'$, this places $(\mathbb{i}_{\mathsf{com}}, \mathsf{cm}, (\boldsymbol{m}^*, \mathsf{td}^*))$ in $\tilde{\mathcal{R}}^h_{\mathsf{com}}$, and $\boldsymbol{m}^*[Q] = \boldsymbol{a}$ because the run is outside $B_3$. Thus the extracted witness is valid for the input query relation.

**Bounds.** Applying Lemma 4.12(ii) to the top commitment before the challenge and its eventual opening gives $\Pr[B_1] \le \kappa_{\mathsf{MT}}(\sigma, b, q)$.

For $B_2$, note that the list $\Lambda(\mathcal{C}, \boldsymbol{c}^\circ, \delta)$ has at most $|\Lambda(\mathcal{C}, \delta)|$ elements, each the encoding of one message, and that the list and every $p_{\boldsymbol{y}'}$ are fixed before $\gamma \xleftarrow{\$} \mathbb{F}$ is sampled. A nonzero $p_{\boldsymbol{y}'}$ has degree at most $t \le |Q| \le n$ and vanishes at $\gamma$ with probability at most $n/|\mathbb{F}|$ by Lemma 3.4, so $\Pr[B_2] \le |\Lambda(\mathcal{C}, \delta)| \cdot n/|\mathbb{F}|$. For $B_3$, apply the VC multi-extraction guarantee of Section 4.2 to $\mathsf{MT}_{\mathsf{btm}}$, with the Merkle bound of Section 4.3.

The reduction adversary's commitment phase runs both stages of $\mathcal{A}$, sampling $\gamma$ itself, and then outputs $\mathsf{rt}_{v_1}, \ldots, \mathsf{rt}_{v_t}$. Here $t \le |Q| \le n$ (pad with fixed roots if needed). All roots are output after all of its queries, so the string extracted for the $i$-th root is $\boldsymbol{m}^*_{v_i}$. Its opening phase outputs $(i, S_{v_i}, \boldsymbol{a}[v_i \,\|\, S_{v_i}], \mathsf{pf}_i)$ for the least $i$ with $\boldsymbol{m}^*_{v_i}[S_{v_i}] \neq \boldsymbol{a}[v_i \,\|\, S_{v_i}]$, which in $B_3$ verifies because $\mathsf{Rdx}.\mathbf{V}$ accepted it. Hence $\Pr[B_3] \le \kappa_{\mathsf{MT}}(\sigma, 2^d, q, n)$, and the three bounds sum to the claim. □

**Honest commitment binding.**

*Proof of Lemma 6.8.* Fix any verifier challenge vector and a $q$-query adversary in the experiment of Definition 4.8. Write $\boldsymbol{m}_{\mathsf{hon}}$ for the honest message and $\boldsymbol{y}_{\mathsf{hon}}$ and $\boldsymbol{c}_{\mathsf{hon}} = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y}_{\mathsf{hon}})$ for its canopy message and top codeword. For a well-formed output $(\boldsymbol{m}, \mathsf{td})$ of the adversary, check the Merkle openings in the relaxed commitment relation: the top opening on $\mathrm{Dom}(\boldsymbol{c})$ and each nonempty block opening on $\mathrm{Dom}(\boldsymbol{m}_v)$. The honest commitment makes at most $2\ell + 2b$ queries, and these checks make at most $b(\log b + 1) + \ell(d+1)$ queries. Thus the combined trace contains at most $q_{\mathsf{hon}} \coloneqq q + \ell(d+3) + b(\log b + 3)$ queries, with collision probability at most $\binom{q_{\mathsf{hon}}}{2}/2^{\sigma}$.

Suppose that the adversary's witness is valid and the combined trace contains no collision. By the path comparison in the proof of Lemma 4.13, the top opening gives $\boldsymbol{c} \preceq \boldsymbol{c}_{\mathsf{hon}}$. Consequently,

$$\delta\big(\boldsymbol{c}_{\mathsf{hon}}, \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y})\big) \le \delta\big(\boldsymbol{c}, \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{y})\big) < \delta \le \delta_{\mathcal{C}} \ .$$

Both full words are codewords, so $\boldsymbol{y} = \boldsymbol{y}_{\mathsf{hon}}$. For every nonempty block of $\boldsymbol{m}$, the relaxed relation and injectivity of $\mathsf{H2F}$ therefore force its root to equal the corresponding honest block root. Comparing its opening paths with the honest block tree gives $\boldsymbol{m}_v \preceq (\boldsymbol{m}_{\mathsf{hon}})_v$. Hence $\boldsymbol{m} \preceq \boldsymbol{m}_{\mathsf{hon}}$, and binding failure implies a collision in the combined trace. □

# C Properties of RBRKS interactive reductions

In this appendix, we prove three properties of round-by-round knowledge-sound interactive reductions (Definition 3.17). These properties let us compose the interactive reductions of Section 5 and then remove the interaction while preserving straight-line knowledge extraction, as required in Sections 7 and 8.

1. An RBRKS interactive reduction has straight-line state-restoration knowledge soundness (Lemma C.4), following [CY24, Theorem 31.3.1] for public-coin IOPs. We adapt the state-restoration notion from Arc [BMNW25] and compare the definitions in Remark C.3.
2. The sequential composition of two RBRKS interactive reductions is RBRKS (Lemma C.5), the analogue of [BMMS26a, Lemma 4.7] for round-by-round tree extractability.
3. The Fiat–Shamir transformation of a straight-line SRKS interactive reduction has a straight-line knowledge extractor (Lemma C.7). This adapts [BMNW25, Lemma B.9] to our definitions, following [CY24, Theorem 16.1.1].

## C.1 RBRKS implies straight-line SRKS

**Definition C.1** (IR state-restoration game)**.** Let $\mathsf{IR}$ be a $\mathsf{k}_{\mathsf{IR}}$-round interactive reduction and let $\vec{f} = (f_i)_{i\in[\mathsf{k}_{\mathsf{IR}}]}$, where independently $f_i \xleftarrow{\$} \mathcal{U}(R_i)$. The game $\mathsf{SRGame}_{\mathsf{IR}}(\vec{f}, \widetilde{\mathbf{P}}^h_{\mathsf{IR}}(1^\sigma))$ proceeds as follows.

1. Repeat the following until $\widetilde{\mathbf{P}}_{\mathsf{IR}}$ halts.
   (a) Using queries to $h$ and its internal state, $\widetilde{\mathbf{P}}_{\mathsf{IR}}$ chooses a query $(i, \mathbb{i}, \mathbb{x}, (\alpha_j)_{j\in[i]})$.
   (b) For the above query, it receives a response $\rho_i \leftarrow f_i(\mathbb{i}, \mathbb{x}, (\alpha_j)_{j\in[i]})$.
2. $\widetilde{\mathbf{P}}_{\mathsf{IR}}$ returns $(\mathbb{i}, \mathbb{x}, (\alpha_j)_{j\in[\mathsf{k}_{\mathsf{IR}}]}, \mathbb{w}')$ to the game, for some witness $\mathbb{w}'$.
3. The game outputs $(\mathbb{i}, \mathbb{x}, (\alpha_j)_{j\in[\mathsf{k}_{\mathsf{IR}}]}, (\rho_j)_{j\in[\mathsf{k}_{\mathsf{IR}}]}, \mathbb{w}')$, where $\rho_i = f_i(\mathbb{i}, \mathbb{x}, (\alpha_j)_{j\in[i]})$ for every $i \in [\mathsf{k}_{\mathsf{IR}}]$.

We call a query to one of the functions $f_i$ a *move*. The game also outputs a trace $\mathsf{tr}_{\mathsf{sr}}$, the ordered tuple of the query-response pairs of $\widetilde{\mathbf{P}}_{\mathsf{IR}}$ for $h$ and $(f_i)_{i\in[\mathsf{k}_{\mathsf{IR}}]}$, where each entry records which of these oracles was queried. A $q$-query ROM state-restoration prover makes at most $q$ queries in total to $h$ and the functions $(f_i)_i$.

**Definition C.2** (Straight-line state-restoration knowledge soundness)**.** An interactive reduction $\mathsf{IR} = (\mathbf{I}_{\mathsf{IR}}, \mathbf{P}_{\mathsf{IR}}, \mathbf{V}_{\mathsf{IR}})$ from $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_2^{\mathcal{U}}$ has *straight-line state-restoration knowledge error* $\kappa^{\mathsf{IR}}_{\mathsf{sr}}(\sigma, n, q)$ *and extraction time* $\mathsf{et}^{\mathsf{IR}}_{\mathsf{sr}}(\sigma, n, q)$ if there is a deterministic extractor $\mathbf{E}_{\mathsf{IR}}$ that makes no random-oracle queries and runs in time at most $\mathsf{et}^{\mathsf{IR}}_{\mathsf{sr}}(\sigma, n, q)$ such that, for every $q$-query ROM state-restoration prover $\widetilde{\mathbf{P}}_{\mathsf{IR}}$,

$$\Pr\left[\begin{array}{l|l} & h \xleftarrow{\$} \mathcal{U}(\sigma) \\ |(\mathbb{i}, \mathbb{x})| \le n & \forall i \in [\mathsf{k}_{\mathsf{IR}}] : f_i \xleftarrow{\$} \mathcal{U}(R_i) \\ \wedge\ b = 1 & (\mathbb{i}, \mathbb{x}, \vec{\alpha}, \vec{\rho}, \mathbb{w}') \xleftarrow{\mathsf{tr}_{\mathsf{sr}}} \mathsf{SRGame}_{\mathsf{IR}}(\vec{f}, \widetilde{\mathbf{P}}^h_{\mathsf{IR}}(1^\sigma)) \\ \wedge\ (\mathbb{i}', \mathbb{x}', \mathbb{w}') \in \tilde{\mathcal{R}}_2^h & (\mathbb{i}', \mathsf{pk}, \mathsf{vk}) \xleftarrow{\mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}}} \mathbf{I}^h_{\mathsf{IR}}(\mathbb{i}) \\ \wedge\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \tilde{\mathcal{R}}_1^h & (b, \mathbb{x}') \xleftarrow{\mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}}} \mathbf{V}^h_{\mathsf{IR}}(\mathsf{vk}, \mathbb{x}; \vec{\alpha}, \vec{\rho}) \\ & \mathbb{w} \leftarrow \mathbf{E}_{\mathsf{IR}}(\mathbb{i}, \mathbb{x}, \vec{\alpha}, \vec{\rho}, \mathbb{i}', \mathbb{x}', \mathbb{w}', \mathsf{tr}_{\mathsf{sr}}, \mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}}, \mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}}) \end{array}\right] \le \kappa^{\mathsf{IR}}_{\mathsf{sr}}(\sigma, n, q) \ .$$

**Remark C.3** (Comparison with the definition in Arc)**.** Arc [BMNW25, Definition B.4] defines straight-line state-restoration knowledge soundness for IORs. The extractor in Definition C.2 receives the same inputs as Arc's extractor for interactive reductions [BMNW25, Lemma B.6], together with the challenges, output statement, and indexer and verifier traces required by our round-by-round extractor. Arc's extractor satisfies this definition by ignoring these additional inputs, with its error evaluated at move and query budgets $q$ and statements of size at most $n$.

**Lemma C.4** (RBRKS implies straight-line SRKS). *Let* $\mathsf{IR} = (\mathbf{I}_{\mathsf{IR}}, \mathbf{P}_{\mathsf{IR}}, \mathbf{V}_{\mathsf{IR}})$ *be a* $\mathsf{k}_{\mathsf{IR}}$*-round interactive reduction from* $\mathcal{R}_1^{\mathcal{U}}$ *to* $\mathcal{R}_2^{\mathcal{U}}$ *with round-by-round knowledge soundness errors* $(\kappa_i(\sigma,n,q))_{i\in[\mathsf{k}_{\mathsf{IR}}]}$ *and extraction times* $(\mathsf{et}_i(\sigma,n,q))_{i\in[\mathsf{k}_{\mathsf{IR}}]}$ *(Definition 3.17). For statements of size at most* $n$*, let* $q_{\mathbf{I}_{\mathsf{IR}}}(n)$ *and* $T_{\mathbf{I}_{\mathsf{IR}}}(n)$ *bound the number of random-oracle queries and the running time of* $\mathbf{I}_{\mathsf{IR}}$*. Let* $q_{\mathbf{V}_{\mathsf{IR}}}(n)$ *and* $T_{\mathbf{V}_{\mathsf{IR}}}(n)$ *be the corresponding bounds for* $\mathbf{V}_{\mathsf{IR}}$*, taken over all such statements and all transcripts. We assume that* $\mathbf{V}_{\mathsf{IR}}$ *rejects an over-long prover message on receipt, without further queries, so that these bounds exist. Let* $q_* := q + q_{\mathbf{I}_{\mathsf{IR}}}(n) + q_{\mathbf{V}_{\mathsf{IR}}}(n)$*. Then* $\mathsf{IR}$ *has straight-line state-restoration knowledge error*

$$\kappa_{\mathsf{sr}}^{\mathsf{IR}}(\sigma,n,q) \le (q+1)\cdot \sum_{i\in[\mathsf{k}_{\mathsf{IR}}]} \kappa_i(\sigma,n,q_*)$$

*and extraction time*

$$\mathsf{et}_{\mathsf{sr}}^{\mathsf{IR}}(\sigma,n,q) \le \sum_{i\in[\mathsf{k}_{\mathsf{IR}}]} \mathsf{et}_i(\sigma,n,q_*) + O\big(T_{\mathbf{I}_{\mathsf{IR}}}(n) + T_{\mathbf{V}_{\mathsf{IR}}}(n)\big) \ .$$

*Proof.* We adapt the proof of [CY24, Theorem 31.3.1] to the knowledge states and trace-based extractors of Definition 3.17. The extractor works backwards from the output witness; a failed extraction identifies a round and one of at most $q+1$ distinct queries to that round's state-restoration oracle.

Let $\mathsf{KState}^h$ and $\mathcal{E}_{\text{rbr}}$ be the knowledge state function and the extractor of Definition 3.17 for $\mathsf{IR}$. Fix a $q$-query state-restoration prover $\widetilde{\mathbf{P}}_{\mathsf{IR}}$ and consider the experiment of Definition C.2 with its notation. For $i \in [\mathsf{k}_{\mathsf{IR}}]$, let $\mathsf{tr}_i := (\alpha_1, \rho_1, \ldots, \alpha_i, \rho_i)$ be the output transcript through the $i$-th challenge, and let $\mathsf{tr}_0 := \varnothing$. Write $\mathsf{tr}_i^- := \mathsf{tr}_{i-1} \,\|\, \alpha_i$ for the prefix immediately before that challenge. We call $(i, \mathbb{i}, \mathbb{x}, (\alpha_j)_{j\in[i]})$ the *$i$-th output query*, whether or not $\widetilde{\mathbf{P}}_{\mathsf{IR}}$ made it, and let $\mathsf{tr}_{\mathsf{sr}}^+$ be $\mathsf{tr}_{\mathsf{sr}}$ followed by the $\mathsf{k}_{\mathsf{IR}}$ output queries with their responses $\rho_i$. There are at most $q+1$ distinct queries to each $f_i$ in $\mathsf{tr}_{\mathsf{sr}}^+$. Let $\mathsf{qtr}_{0,i}$ and $\mathsf{qtr}_{1,i}$ be the base-oracle portions of $\mathsf{tr}_{\mathsf{sr}}$ before and after the first occurrence of the $i$-th output query, respectively.

**Extractor.** The extractor $\mathbf{E}_{\mathsf{IR}}$ replays the indexer and verifier to recover $\mathsf{vk}$ and the verifier trace $\mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}}^{(i)}$ on each prefix $\mathsf{tr}_i$. It sets $\mathbb{w}_{\mathsf{k}_{\mathsf{IR}}} := \mathbb{w}'$, computes for $i = \mathsf{k}_{\mathsf{IR}}, \ldots, 1$

$$\mathbb{w}_{i-1} := \mathcal{E}_{\text{rbr}}\big(\mathbb{i}, \mathbb{x}, \mathsf{qtr}_{0,i}, \mathsf{qtr}_{1,i} \,\|\, \mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}} \,\|\, \mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}}, \mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}}, \mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}}^{(i)}, \mathsf{tr}_i, \mathbb{w}_i\big) \ ,$$

and outputs $\mathbb{w} := \mathbb{w}_0$. The replays take time $O(T_{\mathbf{I}_{\mathsf{IR}}}(n) + T_{\mathbf{V}_{\mathsf{IR}}}(n))$, which gives the claimed extraction time.

**A failing round exists.** If extraction fails, the knowledge state is 1 on $(\mathsf{tr}_{\mathsf{k}_{\mathsf{IR}}}, \mathbb{w}_{\mathsf{k}_{\mathsf{IR}}})$ and 0 on $(\mathsf{tr}_0, \mathbb{w}_0)$. By the prover-moves property, some round $i$ therefore has state 0 on $(\mathsf{tr}_i^-, \mathbb{w}_{i-1})$ and 1 on $(\mathsf{tr}_i, \mathbb{w}_i)$. For $i \in [\mathsf{k}_{\mathsf{IR}}]$ and $j \in [q+1]$, let $X_{i,j}$ be the event that $|(\mathbb{i}, \mathbb{x})| \le n$, the first occurrence of the $i$-th output query in $\mathsf{tr}_{\mathsf{sr}}^+$ is the $j$-th distinct query to $f_i$, $\mathsf{KState}^h(\mathbb{i}, \mathbb{x}, \mathsf{tr}_i^-, \mathbb{w}_{i-1}) = 0$, and $\mathsf{KState}^h(\mathbb{i}, \mathbb{x}, \mathsf{tr}_i, \mathbb{w}_i) = 1$. The failure event is contained in the union of the events $X_{i,j}$, so it suffices to show $\Pr[X_{i,j}] \le \kappa_i(\sigma, n, q_*)$ for every $i$ and $j$.

**Reduction to round-by-round knowledge soundness.** Fix $i$ and $j$. The round-$i$ adversary $\mathcal{A}_{i,j}$ simulates the state-restoration game, including its output queries, forwarding base-oracle queries and sampling $(f_l)_l$ lazily. When it reaches the $j$-th distinct query to $f_i$, say $(i, \mathbb{i}^\dagger, \mathbb{x}^\dagger, (\alpha_m^\dagger)_{m\in[i]})$, it suspends the simulation. For each $l < i$, it reads the response to $(l, \mathbb{i}^\dagger, \mathbb{x}^\dagger, (\alpha_m^\dagger)_{m\in[l]})$ from the table, sampling it if necessary. These responses determine a prefix $\mathsf{tr}^\dagger = (\alpha_1^\dagger, \rho_1^\dagger, \ldots, \alpha_i^\dagger)$, and the adversary outputs $(\mathbb{i}^\dagger, \mathbb{x}^\dagger, \mathsf{tr}^\dagger)$. On receiving the challenge $\rho_i$ of the experiment, it records $\rho_i$ as the response to the suspended query and resumes the simulation. If the simulated output $(\mathbb{i}^\circ, \mathbb{x}^\circ, \vec{\alpha}^\circ, \mathbb{w}^\circ)$ has $|(\mathbb{i}^\circ, \mathbb{x}^\circ)| \le n$ and the $i$-th output query of the simulated game is the suspended query, it runs $\mathbf{I}_{\mathsf{IR}}^h(\mathbb{i}^\circ)$ and $\mathbf{V}_{\mathsf{IR}}^h(\mathsf{vk}^\circ, \mathbb{x}^\circ; \vec{\alpha}^\circ, \vec{\rho}^\circ)$ with the challenges from the table. It runs the

backwards extraction through rounds $\mathsf{k}_{\mathsf{IR}}, \ldots, i+1$ and outputs the resulting $\mathbb{w}_i^\circ$. It makes at most $q_*$ queries to $h$, since it runs the indexer and the verifier only on statements of size at most $n$.

Lazy sampling preserves the joint distribution, including when the adversary fills in earlier-round challenges for $\mathsf{tr}^\dagger$. Couple the experiments using the same base oracle, prover coins, and functions $(f_l)_l$. On $X_{i,j}$, the suspended query is the $i$-th output query, so $\mathcal{A}_{i,j}$ supplies $(\mathbb{i}, \mathbb{x}, \mathsf{tr}_i^-)$ and the witness $\mathbb{w}_i$. The traces of $\mathcal{A}_{i,j}$ before and after the challenge are $\mathsf{qtr}_{0,i}$ and $\mathsf{qtr}_{1,i} \,\|\, \mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}} \,\|\, \mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}}$, and the indexer and verifier traces of the experiment are $\mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}}$ and $\mathsf{qtr}^{(i)}_{\mathbf{V}_{\mathsf{IR}}}$. Thus $\mathcal{E}_{\text{rbr}}$ computes the same $\mathbb{w}_{i-1}$, and $X_{i,j}$ implies a failure of round-by-round knowledge soundness. Consequently $\Pr[X_{i,j}] \le \kappa_i(\sigma, n, q_*)$; a union bound over $i$ and $j$ completes the proof. □

## C.2 Composition of RBRKS interactive reductions

**Sequential composition.** Let $\mathsf{IR}_1 = (\mathbf{I}_1, \mathbf{P}_1, \mathbf{V}_1)$ be a $\mathsf{k}_{\mathsf{IR}_1}$-round interactive reduction from $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_2^{\mathcal{U}}$ and let $\mathsf{IR}_2 = (\mathbf{I}_2, \mathbf{P}_2, \mathbf{V}_2)$ be a $\mathsf{k}_{\mathsf{IR}_2}$-round interactive reduction from $\mathcal{R}_2^{\mathcal{U}}$ to $\mathcal{R}_3^{\mathcal{U}}$, where all three relations are indexed promise relations relative to the same random oracle. The *sequential composition* $\mathsf{IR}_1 \circ \mathsf{IR}_2 = (\mathbf{I}_{\mathsf{IR}}, \mathbf{P}_{\mathsf{IR}}, \mathbf{V}_{\mathsf{IR}})$ is the $(\mathsf{k}_{\mathsf{IR}_1} + \mathsf{k}_{\mathsf{IR}_2})$-round interactive reduction from $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_3^{\mathcal{U}}$ that runs $\mathsf{IR}_1$ and then runs $\mathsf{IR}_2$ on the statement and witness that $\mathsf{IR}_1$ outputs. Its challenge sets and prover-message bounds are those of $\mathsf{IR}_1$ followed by those of $\mathsf{IR}_2$.

- $\mathbf{I}^h_{\mathsf{IR}}(\mathbb{i}_1)$ computes

$$(\mathbb{i}_2, \mathsf{pk}_1, \mathsf{vk}_1) \leftarrow \mathbf{I}_1^h(\mathbb{i}_1) \qquad \text{and} \qquad (\mathbb{i}_3, \mathsf{pk}_2, \mathsf{vk}_2) \leftarrow \mathbf{I}_2^h(\mathbb{i}_2) \ ,$$

  and outputs $(\mathbb{i}_3, (\mathsf{pk}_1, \mathsf{vk}_1, \mathsf{pk}_2), (\mathsf{vk}_1, \mathsf{vk}_2))$.
- $\mathbf{P}^h_{\mathsf{IR}}((\mathsf{pk}_1, \mathsf{vk}_1, \mathsf{pk}_2), \mathbb{x}_1, \mathbb{w}_1)$ and $\mathbf{V}^h_{\mathsf{IR}}((\mathsf{vk}_1, \mathsf{vk}_2), \mathbb{x}_1)$ first run $\langle \mathbf{P}_1^h(\mathsf{pk}_1, \mathbb{x}_1, \mathbb{w}_1), \mathbf{V}_1^h(\mathsf{vk}_1, \mathbb{x}_1)\rangle$ for $\mathsf{k}_{\mathsf{IR}_1}$ rounds, which produces $(b_1, \mathbb{x}_2, \mathbb{w}_2)$. The verifier completes $\mathbf{V}_1$'s output computation before sending its last challenge, so the first-stage trace includes this computation. The prover obtains $(b_1, \mathbb{x}_2)$ by running $\mathbf{V}_1^h(\mathsf{vk}_1, \mathbb{x}_1; \mathsf{tr}_1)$ on the completed first-stage transcript $\mathsf{tr}_1$. In the remaining $\mathsf{k}_{\mathsf{IR}_2}$ rounds, if $b_1 = 1$, then $\mathbf{P}_{\mathsf{IR}}$ runs $\mathbf{P}_2^h(\mathsf{pk}_2, \mathbb{x}_2, \mathbb{w}_2)$ and outputs its witness $\mathbb{w}_3$, and $\mathbf{V}_{\mathsf{IR}}$ runs $\mathbf{V}_2^h(\mathsf{vk}_2, \mathbb{x}_2)$ and outputs the pair $(b_2, \mathbb{x}_3)$ that it produces. If $b_1 = 0$, then $\mathbf{P}_{\mathsf{IR}}$ sends empty messages and outputs $\perp$, and $\mathbf{V}_{\mathsf{IR}}$ sends uniformly random challenges from the sets prescribed by $\mathsf{IR}_2$, makes no random-oracle queries, and outputs $(0, \perp)$.

We call the first $\mathsf{k}_{\mathsf{IR}_1}$ rounds of a transcript its *first-stage part* and the rest its *second-stage part*. Completeness of $\mathsf{IR}_1 \circ \mathsf{IR}_2$ follows from completeness of $\mathsf{IR}_1$ and $\mathsf{IR}_2$, since the prover and the verifier compute the same $(b_1, \mathbb{x}_2)$. Since an interactive argument for $\mathcal{R}_1^{\mathcal{U}}$ in the random oracle model is an interactive reduction from $\mathcal{R}_1^{\mathcal{U}}$ to the trivial relation $\{(\perp, \mathsf{accept}, \perp)\}$, whose relaxed relation is itself, the composition of an interactive reduction with an argument, as in Section 7, is a special case of the above.

**Lemma C.5** (Composition of RBRKS interactive reductions)**.** *Let $\mathsf{IR}_1$ be a $\mathsf{k}_{\mathsf{IR}_1}$-round interactive reduction from $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_2^{\mathcal{U}}$ with round-by-round knowledge soundness errors $(\kappa_{1,i}(\sigma, n, q))_{i \in [\mathsf{k}_{\mathsf{IR}_1}]}$ and extraction times $(\mathsf{et}_{1,i}(\sigma, n, q))_{i \in [\mathsf{k}_{\mathsf{IR}_1}]}$. Let $\mathsf{IR}_2$ be a $\mathsf{k}_{\mathsf{IR}_2}$-round interactive reduction from $\mathcal{R}_2^{\mathcal{U}}$ to $\mathcal{R}_3^{\mathcal{U}}$ with round-by-round knowledge soundness errors $(\kappa_{2,i}(\sigma, n, q))_{i \in [\mathsf{k}_{\mathsf{IR}_2}]}$ and extraction times $(\mathsf{et}_{2,i}(\sigma, n, q))_{i \in [\mathsf{k}_{\mathsf{IR}_2}]}$. For statements of size at most $n$ and arbitrary first-stage transcripts, let $q_1(n)$ and $T_1(n)$ bound the total number of random-oracle queries and the total running time of one execution each of $\mathbf{I}_1$ and $\mathbf{V}_1$. Let $n_2(n)$ bound the size $|(\mathbb{i}_2, \mathbb{x}_2)|$ of the resulting intermediate statement. We assume that $\mathbf{V}_1$ rejects an over-long prover message on receipt, without further queries, so that the three bounds exist. Then $\mathsf{IR}_1 \circ \mathsf{IR}_2$ is a $(\mathsf{k}_{\mathsf{IR}_1} + \mathsf{k}_{\mathsf{IR}_2})$-round*

*interactive reduction from $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_3^{\mathcal{U}}$ with round-by-round knowledge soundness errors*

$$\kappa_r(\sigma,n,q)=\begin{cases}\kappa_{1,r}(\sigma,n,q) & \text{if } r\in[\mathsf{k}_{\mathsf{IR}_1}],\\ \kappa_{2,r-\mathsf{k}_{\mathsf{IR}_1}}(\sigma,n_2(n),q+q_1(n)) & \text{if } r\in\{\mathsf{k}_{\mathsf{IR}_1}+1,\dots,\mathsf{k}_{\mathsf{IR}_1}+\mathsf{k}_{\mathsf{IR}_2}\},\end{cases}$$

*and extraction times*

$$\mathsf{et}_r(\sigma,n,q)=\begin{cases}\mathsf{et}_{1,r}(\sigma,n,q)+O(T_1(n)) & \text{if } r\in[\mathsf{k}_{\mathsf{IR}_1}],\\ \mathsf{et}_{2,r-\mathsf{k}_{\mathsf{IR}_1}}(\sigma,n_2(n),q+q_1(n))+O(T_1(n)) & \text{if } r\in\{\mathsf{k}_{\mathsf{IR}_1}+1,\dots,\mathsf{k}_{\mathsf{IR}_1}+\mathsf{k}_{\mathsf{IR}_2}\}.\end{cases}$$

*Proof.* The construction follows [BMMS26a, Lemma 4.7], which proves composition for round-by-round tree extractability. We use the knowledge state and extractor of $\mathsf{IR}_1$ in the first stage and those of $\mathsf{IR}_2$ in the second stage.

Let $(\mathsf{KState}_1^h,\mathcal{E}_{\text{rbr}1})$ and $(\mathsf{KState}_2^h,\mathcal{E}_{\text{rbr}2})$ be the knowledge state functions and extractors of Definition 3.17 for $\mathsf{IR}_1$ and $\mathsf{IR}_2$, and write $\mathsf{IR}_1\circ\mathsf{IR}_2=(\mathbf{I}_{\mathsf{IR}},\mathbf{P}_{\mathsf{IR}},\mathbf{V}_{\mathsf{IR}})$. Fix $h$ and a statement $(\mathbb{i}_1,\mathbb{x}_1)$, and let $(\mathbb{i}_2,\mathsf{pk}_1,\mathsf{vk}_1):=\mathbf{I}_1^h(\mathbb{i}_1)$ and $(\mathbb{i}_3,\mathsf{pk}_2,\mathsf{vk}_2):=\mathbf{I}_2^h(\mathbb{i}_2)$. For a transcript prefix $\mathsf{tr}=\mathsf{tr}_1\,\|\,\mathsf{tr}_2$ of $\mathsf{IR}_1\circ\mathsf{IR}_2$ whose first-stage part $\mathsf{tr}_1$ is full, let $(b_1,\mathbb{x}_2):=\mathbf{V}_1^h(\mathsf{vk}_1,\mathbb{x}_1;\mathsf{tr}_1)$.

**Knowledge state function.** If the second-stage part of $\mathsf{tr}$ is empty, define

$$\mathsf{KState}^h(\mathbb{i}_1,\mathbb{x}_1,\mathsf{tr},\mathsf{w}):=\mathsf{KState}_1^h(\mathbb{i}_1,\mathbb{x}_1,\mathsf{tr},\mathsf{w})\ .$$

Otherwise, set this value to 0 when $b_1=0$, and to $\mathsf{KState}_2^h(\mathbb{i}_2,\mathbb{x}_2,\mathsf{tr}_2,\mathsf{w})$ when $b_1=1$. Note that the two clauses agree on a full first-stage transcript, by the full-transcript property of $\mathsf{KState}_1$ and the empty-transcript property of $\mathsf{KState}_2$. The knowledge-state properties therefore follow from those of the two components.

**Extractor.** In round $r$, the extractor replays $\mathbf{I}_1(\mathbb{i}_1)$ to recover $(\mathbb{i}_2,\mathsf{pk}_1,\mathsf{vk}_1)$ and split the indexer trace into $\mathsf{qtr}_{\mathbf{I}_1}\,\|\,\mathsf{qtr}_{\mathbf{I}_2}$. If $r\le\mathsf{k}_{\mathsf{IR}_1}$, it outputs $\mathcal{E}_{\text{rbr}1}(\mathbb{i}_1,\mathbb{x}_1,\mathsf{qtr}_0,\mathsf{qtr}_1,\mathsf{qtr}_{\mathbf{I}_1},\mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}},\mathsf{tr}\,\|\,\rho_r,\mathsf{w}')$. If $r=\mathsf{k}_{\mathsf{IR}_1}+s$ with $s\in[\mathsf{k}_{\mathsf{IR}_2}]$, it also replays $\mathbf{V}_1(\mathsf{vk}_1,\mathbb{x}_1;\mathsf{tr}_1)$ to recover $(b_1,\mathbb{x}_2)$ and split the verifier trace into $\mathsf{qtr}_{\mathbf{V}_1}\,\|\,\mathsf{qtr}_{\mathbf{V}_2}$. It outputs $\mathsf{w}'$ if $b_1=0$, and otherwise

$$\mathcal{E}_{\text{rbr}2}\big(\mathbb{i}_2,\mathbb{x}_2,\mathsf{qtr}_0\,\|\,\mathsf{qtr}_{\mathbf{I}_1}\,\|\,\mathsf{qtr}_{\mathbf{V}_1},\mathsf{qtr}_1,\mathsf{qtr}_{\mathbf{I}_2},\mathsf{qtr}_{\mathbf{V}_2},\mathsf{tr}_2\,\|\,\rho_r,\mathsf{w}'\big)\ .$$

The replays cost $O(T_1(n))$, giving the stated extraction times.

**Rounds of the first stage.** A $q$-query adversary against round $r\in[\mathsf{k}_{\mathsf{IR}_1}]$ is also an adversary against the same round of $\mathsf{IR}_1$. The knowledge states and inputs to $\mathcal{E}_{\text{rbr}1}$ agree, so the failure probability is at most $\kappa_{1,r}(\sigma,n,q)$.

**Rounds of the second stage.** Fix $s\in[\mathsf{k}_{\mathsf{IR}_2}]$, let $r:=\mathsf{k}_{\mathsf{IR}_1}+s$, and fix a $q$-query adversary $\mathcal{A}$ against round $r$ of $\mathsf{IR}_1\circ\mathsf{IR}_2$. The adversary $\mathcal{A}_2$ against round $s$ of $\mathsf{IR}_2$ runs $\mathcal{A}$, forwarding its random-oracle queries, to obtain $(\mathbb{i}_1,\mathbb{x}_1,\mathsf{tr}_1\,\|\,\mathsf{tr}_2)$. If $|(\mathbb{i}_1,\mathbb{x}_1)|\le n$, it runs $\mathbf{I}_1^h(\mathbb{i}_1)$ and $\mathbf{V}_1^h(\mathsf{vk}_1,\mathbb{x}_1;\mathsf{tr}_1)$, and if moreover $b_1=1$, it outputs $(\mathbb{i}_2,\mathbb{x}_2,\mathsf{tr}_2)$, a statement of size at most $n_2(n)$ with a round-$s$ prefix of $\mathsf{IR}_2$. On receiving a challenge, it forwards the challenge to $\mathcal{A}$ and outputs the witness that $\mathcal{A}$ returns. It makes at most $q+q_1(n)$ queries, since it runs $\mathbf{I}_1$ and $\mathbf{V}_1$ only on statements of size at most $n$. Couple both experiments using the same oracle, adversary coins, and challenge. A failure for the composition requires $b_1=1$. In this case, $\mathcal{A}_2$ has traces $\mathsf{qtr}_0\,\|\,\mathsf{qtr}_{\mathbf{I}_1}\,\|\,\mathsf{qtr}_{\mathbf{V}_1}$ and $\mathsf{qtr}_1$, so $\mathcal{E}_{\text{rbr}2}$ receives exactly the inputs used by the composed extractor. The two experiments extract the same witness, and the definition of $\mathsf{KState}$ turns a failure for the composition into a failure for $\mathsf{IR}_2$. Its probability is therefore at most $\kappa_{2,s}(\sigma,n_2(n),q+q_1(n))$. □

## C.3 Fiat–Shamir transformation of straight-line SRKS reductions

**Non-interactive reductions.** A *non-interactive reduction* from $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_2^{\mathcal{U}}$ in the random oracle model is a tuple of polynomial-time oracle algorithms $\mathsf{RDX} = (\mathbf{I}_{\mathsf{NI}}, \mathbf{P}_{\mathsf{NI}}, \mathbf{V}_{\mathsf{NI}})$ with the following syntax. The indexer $\mathbf{I}_{\mathsf{NI}}^h(\mathbb{i})$ is deterministic given $h$ and outputs an index $\mathbb{i}'$ for $\mathcal{R}_2$, a prover key $\mathsf{pk}$, and a verifier key $\mathsf{vk}$. The prover $\mathbf{P}_{\mathsf{NI}}^h(\mathsf{pk}, \mathbb{x}, \mathbb{w})$ outputs a proof $\mathsf{pf}$, a new instance $\mathbb{x}'$ and a new witness $\mathbb{w}'$. The verifier $\mathbf{V}_{\mathsf{NI}}^h(\mathsf{vk}, \mathbb{x}, \mathsf{pf})$ is deterministic given $h$ and outputs a decision $b \in \{0,1\}$ and an instance $\mathbb{x}'$. Completeness is defined as in Definition 3.15, with the proof output by $\mathbf{P}_{\mathsf{NI}}$ in place of the interaction.

**Definition C.6** (Straight-line knowledge soundness of a non-interactive reduction)**.** A non-interactive reduction $\mathsf{RDX} = (\mathbf{I}_{\mathsf{NI}}, \mathbf{P}_{\mathsf{NI}}, \mathbf{V}_{\mathsf{NI}})$ from $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_2^{\mathcal{U}}$ has *straight-line knowledge error* $\kappa^{\mathsf{RDX}}(\sigma, n, q)$ *and extraction time* $\mathsf{et}^{\mathsf{RDX}}(\sigma, n, q)$ if there is a deterministic extractor $\mathbf{E}_{\mathsf{NI}}$ that makes no random-oracle queries and runs in time at most $\mathsf{et}^{\mathsf{RDX}}(\sigma, n, q)$ such that, for every $q$-query ROM adversary $\widetilde{\mathbf{P}}_{\mathsf{NI}}$,

$$\Pr\left[\begin{array}{l|l} |(\mathbb{i}, \mathbb{x})| \le n & \text{the oracles of } \mathsf{RDX}, \text{ among them } h \xleftarrow{\$} \mathcal{U}(\sigma), \text{ are sampled} \\ \wedge\ b = 1 & (\mathbb{i}, \mathbb{x}, \mathsf{pf}, \mathbb{w}') \xleftarrow{\mathsf{qtr}} \widetilde{\mathbf{P}}_{\mathsf{NI}}^h(1^\sigma) \\ \wedge\ (\mathbb{i}', \mathbb{x}', \mathbb{w}') \in \tilde{\mathcal{R}}_2^h & (\mathbb{i}', \mathsf{pk}, \mathsf{vk}) \xleftarrow{\mathsf{qtr}_{\mathbf{I}_{\mathsf{NI}}}} \mathbf{I}_{\mathsf{NI}}^h(\mathbb{i}) \\ \wedge\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \tilde{\mathcal{R}}_1^h & (b, \mathbb{x}') \xleftarrow{\mathsf{qtr}_{\mathbf{V}_{\mathsf{NI}}}} \mathbf{V}_{\mathsf{NI}}^h(\mathsf{vk}, \mathbb{x}, \mathsf{pf}) \\ & \mathbb{w} \leftarrow \mathbf{E}_{\mathsf{NI}}(\mathbb{i}, \mathbb{x}, \mathsf{pf}, \mathbb{i}', \mathbb{x}', \mathbb{w}', \mathsf{qtr}, \mathsf{qtr}_{\mathbf{I}_{\mathsf{NI}}}, \mathsf{qtr}_{\mathbf{V}_{\mathsf{NI}}}) \end{array}\right] \le \kappa^{\mathsf{RDX}}(\sigma, n, q) \ .$$

**Hash-chain Fiat–Shamir.** Let $\mathsf{IR} = (\mathbf{I}_{\mathsf{IR}}, \mathbf{P}_{\mathsf{IR}}, \mathbf{V}_{\mathsf{IR}})$ be a $\mathsf{k}_{\mathsf{IR}}$-round interactive reduction from $\mathcal{R}_1^{\mathcal{U}}$ to $\mathcal{R}_2^{\mathcal{U}}$. Besides the base oracle $h$, which $\mathsf{IR}$ and the relations use, $\mathsf{HCFS}[\mathsf{IR}]$ uses an *index oracle* $h_{\mathsf{idx}} \xleftarrow{\$} \mathcal{U}(\sigma)$ and Fiat–Shamir oracles $h_{\mathsf{FS}}^{(i)} \xleftarrow{\$} \mathcal{U}(R_i)$ for $i \in [\mathsf{k}_{\mathsf{IR}}]$, all independent. We call a query to $h_{\mathsf{idx}}$ an *index query*, and we write a query to a Fiat–Shamir oracle as the tuple it encodes. We assume that $|R_i| \ge 2^\sigma$ for every $i$, replacing smaller challenge sets by $R_i \times \{0,1\}^\sigma$ and ignoring the extra component in $\mathsf{IR}$ if necessary. The *hash-chain Fiat–Shamir transformation* $\mathsf{HCFS}[\mathsf{IR}] = (\mathbf{I}_{\mathsf{NI}}, \mathbf{P}_{\mathsf{NI}}, \mathbf{V}_{\mathsf{NI}})$ is [CY24, Construction 15.1.1] without salts and with an index digest added to the first hash.

- $\mathbf{I}_{\mathsf{NI}}^h(\mathbb{i})$ runs the indexer of $\mathsf{IR}$ and computes an *index digest*:

$$(\mathbb{i}', \mathsf{pk}, \mathsf{vk}) \leftarrow \mathbf{I}_{\mathsf{IR}}^h(\mathbb{i}) \qquad \text{and} \qquad d \coloneqq h_{\mathsf{idx}}(\mathbb{i}) \ .$$

  It outputs $(\mathbb{i}', (d, \mathsf{pk}, \mathsf{vk}), (d, \mathsf{vk}))$.
- $\mathbf{P}_{\mathsf{NI}}^h((d, \mathsf{pk}, \mathsf{vk}), \mathbb{x}, \mathbb{w})$ runs $\mathbf{P}_{\mathsf{IR}}^h(\mathsf{pk}, \mathbb{x}, \mathbb{w})$, answering its $i$-th challenge with

$$\rho_1 \coloneqq h_{\mathsf{FS}}^{(1)}(d, \mathbb{x}, \alpha_1) \qquad \text{and} \qquad \rho_i \coloneqq h_{\mathsf{FS}}^{(i)}(\rho_{i-1}, \alpha_i) \quad \text{for } i > 1 \ .$$

  It sets $\mathsf{pf} \coloneqq (\alpha_1, \ldots, \alpha_{\mathsf{k}_{\mathsf{IR}}})$ and computes $(b, \mathbb{x}') \leftarrow \mathbf{V}_{\mathsf{IR}}^h(\mathsf{vk}, \mathbb{x}; \vec{\alpha}, \vec{\rho})$. It outputs $(\mathsf{pf}, \mathbb{x}', \mathbb{w}')$, where $\mathbb{w}'$ is the witness output by $\mathbf{P}_{\mathsf{IR}}$.
- $\mathbf{V}_{\mathsf{NI}}^h((d, \mathsf{vk}), \mathbb{x}, \mathsf{pf})$ outputs $(0, \perp)$ if $\mathsf{pf}$ does not parse as $\mathsf{k}_{\mathsf{IR}}$ prover messages. Otherwise it recomputes $(\rho_1, \ldots, \rho_{\mathsf{k}_{\mathsf{IR}}})$ from $d$ and $\mathsf{pf}$ as above and outputs $(b, \mathbb{x}') \leftarrow \mathbf{V}_{\mathsf{IR}}^h(\mathsf{vk}, \mathbb{x}; \vec{\alpha}, \vec{\rho})$.

We call $\rho_1, \ldots, \rho_{\mathsf{k}_{\mathsf{IR}}}$ the *chain values* of $(d, \mathbb{x}, \vec{\alpha})$. The digest binds the challenges to the index without requiring the verifier to read the index. Completeness follows from that of $\mathsf{IR}$: the honest prover queries each Fiat–Shamir oracle once, so the chain values have the same distribution as the interactive challenges, and both parties compute the same output instance.

**Lemma C.7** (Fiat–Shamir transformation of straight-line SRKS reductions)**.** *Let* $\mathsf{IR}$ *be a* $\mathsf{k}_{\mathsf{IR}}$*-round interactive reduction from* $\mathcal{R}_1^{\mathcal{U}}$ *to* $\mathcal{R}_2^{\mathcal{U}}$ *with challenge sets satisfying* $|R_i| \geq 2^\sigma$ *and with straight-line SRKS error* $\kappa_{\mathsf{sr}}^{\mathsf{IR}}(\sigma, n, q)$ *and extraction time* $\mathsf{et}_{\mathsf{sr}}^{\mathsf{IR}}(\sigma, n, q)$*. Then* $\mathsf{HCFS}[\mathsf{IR}]$ *has extraction time* $\mathsf{et}_{\mathsf{sr}}^{\mathsf{IR}}(\sigma, n, q)$ *with SRKS error*

$$\kappa^{\mathsf{HCFS}[\mathsf{IR}]}(\sigma, n, q) \leq \kappa_{\mathsf{sr}}^{\mathsf{IR}}(\sigma, n, q) + \frac{(q + \mathsf{k}_{\mathsf{IR}} + 1)^2}{2^\sigma} \quad .$$

*Proof.* We follow the proof of [CY24, Theorem 16.1.1], adapting it to reductions as in [BMNW25, Lemma B.9]. The simulation turns hash queries that extend a known chain into state-restoration moves; collisions and guesses of future chain values account for the additional error.

**State-restoration prover.** The state-restoration prover $\widetilde{\mathbf{P}}_{\mathsf{IR}}$ runs $\widetilde{\mathbf{P}}_{\mathsf{NI}}$, forwards every base-oracle query to $h$, and answers repeated queries to the other oracles consistently. For a new Fiat–Shamir query, it applies the following rules.

- For a query $(d, \mathbb{x}, \alpha_1)$ to $h_{\mathsf{FS}}^{(1)}$, if $d$ is the response to exactly one earlier distinct index query $\mathbb{i}$, it makes the move $(1, \mathbb{i}, \mathbb{x}, \alpha_1)$ and returns its response.
- For a query $(\rho, \alpha_i)$ to $h_{\mathsf{FS}}^{(i)}$ with $i > 1$, if $\rho$ is the response to exactly one earlier move of round $i - 1$, say $(i - 1, \mathbb{i}, \mathbb{x}, (\alpha_j)_{j \in [i-1]})$, it makes the move $(i, \mathbb{i}, \mathbb{x}, (\alpha_j)_{j \in [i]})$ and returns its response.

It answers all other new queries to the additional oracles with independent uniform elements of their respective output sets. On a validly parsed output $(\mathbb{i}, \mathbb{x}, \mathsf{pf}, \mathbb{w}')$, it returns $(\mathbb{i}, \mathbb{x}, \vec{\alpha}, \mathbb{w}')$ to the game.

The simulator makes at most one query or move per adversary query, and distinct new queries give distinct moves. Its responses therefore have the correct distribution, and we couple the experiments using the same base oracle, adversary coins, and query responses.

**The bad event.** Let $\mathsf{qtr}^+$ consist of the index and Fiat–Shamir entries of $\mathsf{qtr}$, the index entry of $\mathsf{qtr}_{\mathbf{I}_{\mathsf{NI}}}$, and the $\mathsf{k}_{\mathsf{IR}}$ Fiat–Shamir entries of $\mathsf{qtr}_{\mathbf{V}_{\mathsf{NI}}}$, in that order. There are at most $q + \mathsf{k}_{\mathsf{IR}} + 1$ entries in total. Define the bad event to be the union of the following events:

- Two distinct queries to the same oracle in $\mathsf{qtr}^+$ receive the same response.
- The response to a query that is new to its oracle equals the digest or preceding chain value in an earlier Fiat–Shamir query.

Each fresh response is uniform on a set of size at least $2^\sigma$ and is compared with at most two fixed values per earlier entry. Thus, writing $N := q + \mathsf{k}_{\mathsf{IR}} + 1$, a union bound gives

$$\Pr[\text{bad event}] \leq 2\binom{N}{2} 2^{-\sigma} \leq \frac{N^2}{2^\sigma} \quad .$$

Outside the bad event, any adversarial query on the output chain follows the query that produced its index digest or preceding chain value, which identifies its predecessor uniquely. Induction along the chain shows that each verifier query either was answered by the corresponding state-restoration move or is fresh, in which case we couple its response to the fresh output move. The chain values then agree with the game's challenges, and both experiments run $\mathbf{I}_{\mathsf{IR}}$ and $\mathbf{V}_{\mathsf{IR}}$ on the same inputs.

**Extractor.** The extractor reconstructs $\mathsf{tr}_{\mathsf{sr}}$ by applying the simulator's rules to $\mathsf{qtr}$, retaining base-oracle entries and the moves made for new Fiat–Shamir queries. It reads the chain values from the verifier trace and restricts the indexer and verifier traces to the base oracle, then outputs

$$\mathbf{E}_{\mathsf{IR}}(\mathbb{i}, \mathbb{x}, \vec{\alpha}, \vec{\rho}, \mathbb{i}', \mathbb{x}', \mathbb{w}', \mathsf{tr}_{\mathsf{sr}}, \mathsf{qtr}_{\mathbf{I}_{\mathsf{IR}}}, \mathsf{qtr}_{\mathbf{V}_{\mathsf{IR}}}) \quad .$$

Outside the bad event, these are precisely the inputs in the state-restoration experiment, so the claimed error bound follows from Definition C.2. The extraction time is that of the single call to $\mathbf{E}_{\mathsf{IR}}$, under our trace-processing convention. □

# D Proving a batch of SEP claims

We construct BatchPrx, an IOPP that proves a batch of $\mathsf{SEP}[\mathcal{C}, \delta]$ claims using a single IOPP for $\mathsf{MEP}[\mathcal{C}, \delta]$. We first recall the coding-theoretic tools used in the construction, then describe the protocol and prove its round-by-round knowledge soundness.

## D.1 Prerequisites

We view a word over the alphabet $\mathbb{F}^\eta$ as a matrix $\boldsymbol{V} \in \mathbb{F}^{\eta \times n}$, with each column representing one symbol. Hamming distance between such words counts the columns in which they differ.

**Definition D.1** (Interleaved codes)**.** Let $\eta \in \mathbb{N}$ be the interleaving parameter and $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$. The $\eta$-interleaved code $\mathcal{C}^\eta \subseteq (\mathbb{F}^\eta)^n$ consists of the matrices whose rows belong to $\mathcal{C}$. It has block length $n$, message length $k$ over the alphabet $\mathbb{F}^\eta$, and relative distance $\delta_{\mathcal{C}}$. Given a message $\boldsymbol{M} \in (\mathbb{F}^\eta)^k$, the encoder $\mathsf{Enc}_{\mathcal{C}^\eta}$ encodes each row of $\boldsymbol{M}$ under $\mathcal{C}$.

The following lemma states that if a code is list-decodable then so is its interleaving.

**Lemma D.2** (List decodability of interleaved codes, [GGR11, Theorem 2.5])**.** *Let $\mathcal{C}$ be a code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$, and let $\delta \in (0, \delta_{\mathcal{C}})$. Let $b := \left\lceil \frac{\delta}{\delta_{\mathcal{C}} - \delta} \right\rceil$ and $r := \left\lceil \log_2 \frac{\delta_{\mathcal{C}}}{\delta_{\mathcal{C}} - \delta} \right\rceil$. For every $\eta \geq 1$, we have*

$$|\Lambda(\mathcal{C}^\eta, \delta)| \leq \binom{b+r}{r} \cdot |\Lambda(\mathcal{C}, \delta)|^r \ .$$

### D.1.1 Mutual correlated agreement

Linear codes exhibit a property called *correlated agreement* used extensively in previous work on IOPPs and accumulation schemes [AHIV17; BBHR18; ACFY24]. A stronger notion known as *mutual correlated agreement* was introduced in [ACFY25].

**Definition D.3** (Mutual correlated agreement for proximity generators)**.** Let $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$, and let $S$ be a finite nonempty seed set. We say that $\mathsf{Gen} : S \to \mathbb{F}^\eta$ has mutual correlated agreement for $\mathcal{C}$ with proximity bound $\mathsf{B}^*$ and error $\mathsf{err}^*$ if, for every $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_\eta \in \mathbb{F}^n$ and $\delta \in (0, 1 - \mathsf{B}^*)$, the following holds:

$$\Pr_{\substack{s \xleftarrow{\$} S \\ (\gamma_1, \ldots, \gamma_\eta) \leftarrow \mathsf{Gen}(s)}} \left[ \exists S \subseteq [n] \quad \text{s.t.} \quad \begin{array}{l} |S| \geq (1-\delta)n \ , \\ \exists \boldsymbol{c} \in \mathcal{C} : \sum_{j \in [\eta]} \gamma_j \cdot \boldsymbol{w}_j[S] = \boldsymbol{c}[S] \ , \\ \exists i \in [\eta], \forall \boldsymbol{c}' \in \mathcal{C} : \boldsymbol{w}_i[S] \neq \boldsymbol{c}'[S] \end{array} \right] \leq \mathsf{err}^* \ .$$

For fixed words $[\boldsymbol{w}_i]_{i \in [\eta]}$ and proximity parameter $\delta$, we call $s$ a *good challenge* if it avoids the bad event in Definition D.3. Thus, whenever the generated linear combination agrees with a codeword on a sufficiently large set, every input word agrees with a codeword on that same set.

**Definition D.4.** We use $\delta_{\mathsf{mca}}(\mathcal{C}) := 1 - \inf \mathsf{B}^*$ to denote the proximity radius for mutual correlated agreement of the code $\mathcal{C}$. The infimum ranges over proximity bounds for which some generator with $\eta \geq 2$ has mutual correlated agreement with negligible error, in the asymptotic regime of the code and field parameters under consideration.

Known results [GKL24; Zei26; BCGM26] give mutual correlated agreement for affine-space generators up to radius $1-(1-\delta_{\mathcal{C}}+\epsilon)^{1/3}$, for $\epsilon \in (0, \delta_{\mathcal{C}})$, with error negligible when the field is sufficiently large relative to the other parameters. This gives the same lower bound on $\delta_{\mathsf{mca}}(\mathcal{C})$. The radius can depend on the generator: the univariate powers generator used below must satisfy the explicit hypothesis of Lemma D.8. For arbitrary linear codes and batch size $\eta \geq 2$, the general bound for that generator in [BCGM26, Theorem 3] reaches radius $1-(1-\delta_{\mathcal{C}}+\epsilon)^{1/(\eta+1)}$.

### D.1.2 Erasure correction

**Definition D.5.** A code $\mathcal{C}$ with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$ supports erasure correction with correction time $\mathsf{ecor}_{\mathcal{C}}$ if there exists a deterministic algorithm $\mathcal{E}_{\mathcal{C}}$ (the erasure corrector) that given as input some $\boldsymbol{w} \in \mathbb{F}^n$ and a set $S \subseteq [n]$, behaves as follows:

- If $|S| > (1-\delta_{\mathcal{C}})n$ and there exists a codeword $\boldsymbol{c} \in \mathcal{C}$ such that $\boldsymbol{c}[S] = \boldsymbol{w}[S]$ (note that such a codeword is unique), then $\mathcal{E}_{\mathcal{C}}(\boldsymbol{w}, S)$ outputs $\boldsymbol{c}$.
- Otherwise, $\mathcal{E}_{\mathcal{C}}(\boldsymbol{w}, S)$ outputs $\perp$.

Moreover, $\mathcal{E}_{\mathcal{C}}$ performs at most $\mathsf{ecor}_{\mathcal{C}}$ field operations.

**Lemma D.6** ([GRS12])**.** *Every linear code $\mathcal{C}$ with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$ supports erasure correction with correction time $O(n^3)$.*

**Lemma D.7.** *Let $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$. Let $\eta \in \mathbb{N}$, $r_1, \ldots, r_\eta \in \mathbb{F}$, and $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_\eta \in \mathbb{F}^n$ be words. Define $\boldsymbol{w}' := \sum_{i=1}^{\eta} r_i \cdot \boldsymbol{w}_i$. Suppose that $S \subseteq [n]$ satisfies $|S| > (1-\delta_{\mathcal{C}})n$, and that codewords $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\eta, \boldsymbol{c}' \in \mathcal{C}$ satisfy $\boldsymbol{c}_i[S] = \boldsymbol{w}_i[S]$ for every $i \in [\eta]$ and $\boldsymbol{c}'[S] = \boldsymbol{w}'[S]$. Then $\boldsymbol{c}' = \sum_{i=1}^{\eta} r_i \cdot \boldsymbol{c}_i$. In particular, if $\boldsymbol{c}_i = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}_i)$ for every $i \in [\eta]$ and $\boldsymbol{c}' = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}')$, then $\boldsymbol{m}' = \sum_{i=1}^{\eta} r_i \cdot \boldsymbol{m}_i$.*

*Proof.* By linearity, $\widetilde{\boldsymbol{c}} := \sum_{i=1}^{\eta} r_i \boldsymbol{c}_i$ is a codeword, and it agrees with $\boldsymbol{c}'$ on $S$. Thus $\delta(\widetilde{\boldsymbol{c}}, \boldsymbol{c}') \leq (n-|S|)/n < \delta_{\mathcal{C}}$, which implies $\widetilde{\boldsymbol{c}} = \boldsymbol{c}'$. The final claim follows from the linearity and injectivity of $\mathsf{Enc}_{\mathcal{C}}$. □

## D.2 The construction of BatchPrx

The protocol has two batching steps. First, a batched sumcheck reduces the $\eta$ input claims to evaluation claims about the underlying messages at a shared point $\boldsymbol{r} \in \mathbb{F}^{\log k}$. The verifier then samples a fresh challenge to combine these evaluation claims, and runs an IOPP for $\mathsf{MEP}[\mathcal{C}, \delta]$ on the corresponding linear combination of the input oracles. Mutual correlated agreement lets the extractor recover all input messages from a witness for this combined claim.

We present the full construction of BatchPrx in Fig. 9 and summarize its properties in Lemma D.8.

**Lemma D.8.** *Let $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$, and let $\mathsf{Prx}$ be a complete $\mathsf{k}_{\mathsf{Prx}}$-round RBRKS IOPP for $\mathsf{MEP}[\mathcal{C}, \delta]$ with round-by-round knowledge soundness errors $\leq 2^{-\lambda}$ and total extraction time $\mathsf{et}_{\mathsf{Prx}}$. Suppose that $\eta \geq 1$ is an integer and that $\mathsf{Gen} : \mathbb{F} \to \mathbb{F}^\eta$, with $\mathsf{Gen}(\gamma) = (1, \gamma, \gamma^2, \ldots, \gamma^{\eta-1})$, has mutual correlated agreement for $\mathcal{C}$ with proximity bound $\mathsf{B}^*$ and error $\mathsf{err}^*$. Let $\delta \in (0, \min\{\delta_{\mathcal{C}}, 1-\mathsf{B}^*\})$. Write $L := \Lambda(\mathcal{C}^\eta, \delta)$. If*

$$\max\left\{\frac{2L}{|\mathbb{F}|}, \frac{(\eta-1)L}{|\mathbb{F}|} + \mathsf{err}^*\right\} \leq 2^{-\lambda} \ ,$$

*then $\mathsf{BatchPrx}$ of Fig. 9 is a complete $(\mathsf{k}_{\mathsf{Prx}} + \log k + 2)$-round RBRKS IOPP for $\mathsf{SEP}[\mathcal{C}, \delta]^\eta$ with round-by-round knowledge soundness errors $\leq 2^{-\lambda}$ and total extraction time*

$$O\big(\eta \cdot n^3 + \eta \cdot n \cdot \mathsf{k}_{\mathsf{Prx}}\big) + \mathsf{et}_{\mathsf{Prx}} \ .$$

*Its knowledge states are strongly monotone whenever those of* Prx *are.*

We defer the proof of Lemma D.8 to Appendix D.3.

**Complexity measures.** The complexity measures of BatchPrx are as follows:

- *Rounds.* BatchPrx has $\mathsf{k}_{\mathsf{Prx}} + \log k + 2$ rounds.
- *Oracle lengths and plaintext communication.* In the first $\log k + 2$ rounds, $\Pi_i$ is empty. The first plaintext message $\alpha_1$ is also empty. Each of the next $\log k$ plaintext messages contains two field elements encoding a quadratic sumcheck polynomial, as specified in Fig. 9. The plaintext message in round $\log k + 2$ contains the $\eta$ claimed evaluations $[\zeta'_i]_{i\in[\eta]}$. Thus, the batching prefix adds $2\log k + \eta$ field elements of plaintext communication; the remaining oracle lengths and plaintext communication are those of Prx.
- *Query complexity.* Suppose that the verifier of Prx makes $q_{\mathsf{Prx},\mathbb{y}}$ queries to its instance oracle and $q_{\mathsf{Prx},\Pi}$ queries to its proof oracles. The verifier of BatchPrx makes $q_{\mathsf{Prx},\mathbb{y}}$ queries to each input oracle $\boldsymbol{w}_i$, for a total of $\eta \cdot q_{\mathsf{Prx},\mathbb{y}}$ input-oracle queries. Its proof-oracle query complexity is $q_{\mathsf{Prx},\Pi}$, and its index-oracle queries are those of Prx. Reading the plaintext messages does not contribute to oracle-query complexity.
- *Prover time.* Given the Boolean evaluation tables of the multilinear polynomials $f_i$, the prover runs the batched sumcheck and computes the claimed evaluations in $O(\eta \cdot k)$ field operations. Forming the combined word $\boldsymbol{w}'$ takes $O(\eta \cdot n)$ additional operations. Thus $t_{\mathsf{BatchPrx}} = t_{\mathsf{Prx}} + O(\eta \cdot n)$, with any cost of constructing the tables of the $f_i$ added separately.

**Indexer input:** $\mathbb{i} = (\mathcal{C}, \delta)^\eta$.
**Indexer output:** The indexer runs the indexer of Prx to obtain $(\iota_{\mathsf{Prx}}, \mathbb{I}_{\mathsf{Prx}}, \mathbb{i}' = \bot) \leftarrow \mathcal{I}_{\mathsf{Prx}}(\mathcal{C}, \delta)$, and outputs

- $\mathbb{i}' := \bot$,
- $\iota := \iota_{\mathsf{Prx}}$, and
- $\mathbb{I} := \mathbb{I}_{\mathsf{Prx}}$.

**Input instance:** $\mathbb{x} = [(f_i, \zeta_i)]_{i\in[\eta]}$.
**Input instance oracle:** $\mathbb{y} = [\boldsymbol{w}_i]_{i\in[\eta]}$.
**Input witness:** $\mathbb{w} = [\boldsymbol{m}_i]_{i\in[\eta]}$.

$\langle \mathcal{P}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathbb{w}), \mathcal{V}^{\mathbb{I},\mathbb{y}}(\iota, \mathbb{x}) \rangle$: Write $\mu := \log k$ and $M_i = (\Pi_i, \alpha_i)$ for the prover's message in round $i$. In the first $\mu + 2$ rounds, $\Pi_i = ()$ and all coefficients and evaluations below are sent in the plaintext component $\alpha_i$.

1. $\mathcal{P}$ sends $M_1 = ((), ())$; $\mathcal{V}$ samples $\beta \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$.
2. Define $\zeta_{\mathsf{sc}} := \sum_{i\in[\eta]} \beta^{i-1} \cdot \zeta_i$ and $g(X_1, \ldots, X_\mu) \in \mathbb{F}^{\leq 2}[X_1, \ldots, X_\mu]$ by
$$g(X_1, \ldots, X_\mu) := \sum_{i\in[\eta]} \beta^{i-1} \cdot f_i(X_1, \ldots, X_\mu, \hat{m}_i(X_1, \ldots, X_\mu)) \ .$$
3. $\mathcal{P}$ and $\mathcal{V}$ run $\mu$ rounds of sumcheck for the claim $\sum_{\boldsymbol{b}\in\{0,1\}^\mu} g(\boldsymbol{b}) = \zeta_{\mathsf{sc}}$, sending the constant and quadratic coefficients of the $j$-th round polynomial in $\alpha_{j+1}$; the linear coefficient is determined by the sumcheck consistency equation. The resulting claim is $g(\boldsymbol{r}) = y_{\boldsymbol{r}}$, where $\boldsymbol{r} = (r_1, \ldots, r_\mu) \in \mathbb{F}^\mu$ and $y_{\boldsymbol{r}}$ is the final running claim. If $\mu = 0$, set $\boldsymbol{r} = ()$ and $y_{\boldsymbol{r}} = \zeta_{\mathsf{sc}}$.

4. $\mathcal{P}$ sends $\alpha_{\mu+2} \coloneqq [\zeta'_i]_{i\in[\eta]}$ to $\mathcal{V}$, where $\zeta'_i \coloneqq \hat{m}_i(\boldsymbol{r})$.
5. $\mathcal{V}$ checks that $\sum_{i\in[\eta]} \beta^{i-1} \cdot f_i(\boldsymbol{r}, \zeta'_i) = y_{\boldsymbol{r}}$, rejecting if the check fails.

   **Batching the evaluation claims**
6. $\mathcal{V}$ samples $\gamma \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$.
7. Define

$$\boldsymbol{w}' \coloneqq \sum_{i\in[\eta]} \gamma^{i-1} \cdot \boldsymbol{w}_i, \quad \boldsymbol{m}' \coloneqq \sum_{i\in[\eta]} \gamma^{i-1} \cdot \boldsymbol{m}_i \quad \text{and} \quad \zeta' \coloneqq \sum_{i\in[\eta]} \gamma^{i-1} \cdot \zeta'_i \ .$$

8. Define the MEP inputs

$$\mathbb{i}_{\mathsf{Prx}} \coloneqq (\mathcal{C}, \delta), \quad \mathbb{x}' \coloneqq (\boldsymbol{r}, \zeta'), \quad \mathbb{y}' \coloneqq \boldsymbol{w}', \quad \mathbb{w}' \coloneqq \boldsymbol{m}' \ .$$

9. Let $(\mathcal{P}_{\mathsf{Prx}}, \mathcal{V}_{\mathsf{Prx}})$ be the prover and verifier of $\mathsf{Prx}$. $\mathcal{P}$ and $\mathcal{V}$ run

$$\left\langle \mathcal{P}_{\mathsf{Prx}}(\mathbb{i}_{\mathsf{Prx}}, \mathbb{x}', \mathbb{y}', \mathbb{w}'), \mathcal{V}_{\mathsf{Prx}}^{\mathbb{I}_{\mathsf{Prx}}, \mathbb{y}'}(\iota_{\mathsf{Prx}}, \mathbb{x}') \right\rangle \ ,$$

   where $\mathcal{V}$ simulates a query to coordinate $j$ of $\boldsymbol{w}'$ by querying coordinate $j$ of every $\boldsymbol{w}_i$ and returning the indicated linear combination. Undefined input coordinates give a simulated answer of $\perp$; malformed plaintext messages in the batching prefix cause rejection. The verifier accepts if $\mathcal{V}_{\mathsf{Prx}}$ accepts and the preceding check passes.

**Figure 9:** BatchPrx, an IOPP for $\mathsf{SEP}[\mathcal{C}, \delta]^\eta$.

## D.3 Proof of Lemma D.8

Completeness of BatchPrx follows from the completeness of the sumcheck protocol and of Prx. For knowledge soundness, we define a knowledge state function and extractor and bound the extraction error at each verifier challenge. Throughout, $(\mathcal{R}, \tilde{\mathcal{R}}) = \mathsf{SEP}[\mathcal{C}, \delta]^\eta$, $\mathbb{i} = (\mathcal{C}, \delta)^\eta$, $\mathbb{x} = [(f_i, \zeta_i)]_{i\in[\eta]}$, and $\mathbb{y} = [\boldsymbol{w}_i]_{i\in[\eta]}$.

Let $\mu \coloneqq \log k$. We denote the round polynomials sent by the prover in the sumcheck protocol by $g_1(X), \ldots, g_\mu(X)$ and the verifier's sumcheck challenges by $r_1, \ldots, r_\mu$. Each prover message is $M_i = (\Pi_i, \alpha_i)$. In the first $\mu + 2$ rounds, $\Pi_i = ()$; the plaintext messages are $\alpha_1 = ()$, the constant and quadratic coefficients of $g_j$ in $\alpha_{j+1}$ for $j \in [\mu]$, and $\alpha_{\mu+2} = [\zeta'_i]_{i\in[\eta]}$. The remaining prover messages are those of Prx: $M_{\mu+2+j} = M_{\mathsf{Prx},j} = (\Pi_{\mathsf{Prx},j}, \alpha_{\mathsf{Prx},j})$. We write $\boldsymbol{r}_0$ for the empty tuple, and use the assumption $\delta \in (0, \min\{\delta_{\mathcal{C}}, 1 - \mathsf{B}^*\})$ throughout. Write $v_0 \coloneqq \sum_{i\in[\eta]} \beta^{i-1}\zeta_i$ and $v_j \coloneqq g_j(r_j)$ for $j \in [\mu]$.

**Knowledge state function and extractor.** We specify states after verifier challenges below. At prover moves in the batching prefix, the preceding state is conjoined with the checks enabled by the new message, including the final sumcheck check after $[\zeta'_i]_{i\in[\eta]}$ is sent. A malformed plaintext message makes all subsequent states 0. During Prx, we inherit its states at prover moves, conjoined with the prefix checks.

1. **Initial transcript.** We set $\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \varnothing, \mathsf{w}) = 1$ if and only if $(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{w}) \in \tilde{\mathcal{R}}$.

2. **Sumcheck batching randomness.** In the first round, the prover sends $M_1 = ((), ())$, and the verifier samples $\beta \xleftarrow{\$} \mathbb{F}$. Before this challenge, the transcript is $\mathsf{tr} = (M_1)$ and the knowledge state equals the initial state.

**Knowledge state function:** We set $\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \beta, \mathsf{w}) = 1$ if and only if $\mathsf{w} = [\boldsymbol{m}_i]_{i\in[\eta]}$ such that the following hold:

(a) $[\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}_i)]_{i\in[\eta]} \in \Lambda\left(\mathcal{C}^{\eta}, [\boldsymbol{w}_i]_{i\in[\eta]}, \delta\right)$.[16]
(b) $\sum_{\boldsymbol{b}\in\{0,1\}^{\mu}} \left(\sum_{i\in[\eta]} \beta^{i-1} \cdot f_i(\boldsymbol{b}, \hat{m}_i(\boldsymbol{b}))\right) = \sum_{i\in[\eta]} \beta^{i-1} \cdot \zeta_i$.

**Extractor:** The extractor returns $\mathsf{w}$ unchanged.

3. **Sumcheck rounds.** Round $j+1$ of $\mathsf{BatchPrx}$ corresponds to sumcheck round $j \in [\mu]$. Before the verifier samples its challenge, the transcript is

$$\mathsf{tr} = (M_1, \beta, (M_{i+1}, r_i)_{i<j}, M_{j+1}) \quad ,$$

and the verifier samples $r_j \xleftarrow{\$} \mathbb{F}$. Define $\boldsymbol{r}_j \coloneqq (r_1, \ldots, r_j)$.

**Knowledge state function:** We set $\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, r_j, \mathsf{w}) = 1$ if and only if $\mathsf{w} = [\boldsymbol{m}_i]_{i\in[\eta]}$ such that the following hold:

(a) $[\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}_i)]_{i\in[\eta]} \in \Lambda\left(\mathcal{C}^{\eta}, [\boldsymbol{w}_i]_{i\in[\eta]}, \delta\right)$.
(b) $g_1(0) + g_1(1) = \sum_{i\in[\eta]} \beta^{i-1} \cdot \zeta_i$, and, for $i < j$, $g_{i+1}(0) + g_{i+1}(1) = g_i(r_i)$.
(c) $\sum_{\boldsymbol{b}'\in\{0,1\}^{\mu-j}} \left(\sum_{i\in[\eta]} \beta^{i-1} \cdot f_i((\boldsymbol{r}_j, \boldsymbol{b}'), \hat{m}_i(\boldsymbol{r}_j, \boldsymbol{b}'))\right) = g_j(r_j)$.

**Extractor:** The extractor returns $\mathsf{w}$ unchanged.

4. **Codeword batching randomness.** Before the challenge in round $\mu+2$, the transcript is

$$\mathsf{tr} = \left(M_1, \beta, (M_{i+1}, r_i)_{i\in[\mu]}, M_{\mu+2}\right) \quad ,$$

and the verifier samples $\gamma \xleftarrow{\$} \mathbb{F}$. Define

$$\boldsymbol{w}' \coloneqq \sum_{i\in[\eta]} \gamma^{i-1} \cdot \boldsymbol{w}_i, \qquad \zeta' \coloneqq \sum_{i\in[\eta]} \gamma^{i-1} \cdot \zeta_i' \quad .$$

**Knowledge state function:** We set $\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \gamma, \mathsf{w}) = 1$ if and only if $\mathsf{w} = \boldsymbol{m}'$ such that the following hold:

(a) $\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}') \in \Lambda(\mathcal{C}, \boldsymbol{w}', \delta)$.
(b) $\hat{m}'(\boldsymbol{r}_\mu) = \zeta'$.
(c) $g_j(0) + g_j(1) = v_{j-1}$ for every $j \in [\mu]$.
(d) $\sum_{i\in[\eta]} \beta^{i-1} \cdot f_i(\boldsymbol{r}_\mu, \zeta_i') = v_\mu$.

The first two conditions state that $((\mathcal{C}, \delta), (\boldsymbol{r}_\mu, \zeta'), \boldsymbol{w}', \boldsymbol{m}')$ lies in the relaxed relation of $\mathsf{MEP}[\mathcal{C}, \delta]$, i.e., that the knowledge state of $\mathsf{Prx}$ on the empty transcript is 1.

**Extractor:** The extractor $\mathcal{E}_{\text{rbr}}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \gamma, \mathsf{w}) \coloneqq \mathsf{Extract}(\boldsymbol{w}', [\boldsymbol{w}_i]_{i\in[\eta]}, \boldsymbol{m}')$ is defined as follows.

$\mathsf{Extract}(\boldsymbol{w}', [\boldsymbol{w}_i]_{i\in[\eta]}, \boldsymbol{m}')$ :

(a) Define $S \coloneqq \{j \in [n] : \boldsymbol{w}'[j] = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}')[j]\}$.
(b) Let $\mathcal{E}_{\mathcal{C}}$ be the erasure corrector given by Lemma D.6.

[16] Here, and in the rest of the proof, we abuse notation and use $[\boldsymbol{w}_i]_{i\in[\eta]}$ to denote the matrix whose $i$-th row is $\boldsymbol{w}_i$.

(c) For each $i \in [\eta]$, compute $\boldsymbol{c}_i \leftarrow \mathcal{E}_{\mathcal{C}}(\boldsymbol{w}_i, S)$. If any call outputs $\perp$, output $\perp$.
(d) Otherwise, output the unique messages $[\boldsymbol{m}_i]_{i\in[\eta]}$ satisfying $\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}_i) = \boldsymbol{c}_i$ for every $i \in [\eta]$.

5. **Rounds corresponding to** $\mathsf{Prx}$**.** Round $\mu + 2 + j$ of $\mathsf{BatchPrx}$ corresponds to round $j \in [\mathsf{k}_{\mathsf{Prx}}]$ of $\mathsf{Prx}$. Just before its verifier challenge, the transcript has the form

$$\mathsf{tr} = \big(M_1, \beta, (M_{i+1}, r_i)_{i\in[\mu]}, M_{\mu+2}, \gamma\big) \,\|\, \mathsf{tr}' \ ,$$

where $\mathsf{tr}' = (M_{\mathsf{Prx},1}, \rho_1, \ldots, M_{\mathsf{Prx},j})$ is the corresponding transcript prefix for $\mathsf{Prx}$, and the verifier is yet to sample $\rho_j$. Define $\boldsymbol{w}'$ and $\zeta'$ as above, and let $\mathbb{i}' \coloneqq (\mathcal{C}, \delta)$, $\mathbb{x}' \coloneqq (\boldsymbol{r}_\mu, \zeta')$, and $\mathbb{y}' \coloneqq \boldsymbol{w}'$ be the $\mathsf{MEP}[\mathcal{C}, \delta]$ statement on which $\mathsf{Prx}$ is run.

**Knowledge state function:** We set $\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \rho_j, \mathsf{w}) = 1$ if and only if the following hold:

(a) $\mathsf{KState}_{\mathsf{Prx}}(\mathbb{i}', \mathbb{x}', \mathbb{y}', \mathsf{tr}' \,\|\, \rho_j, \mathsf{w}) = 1$, where $\mathsf{KState}_{\mathsf{Prx}}$ is the knowledge state function for $\mathsf{Prx}$.
(b) $g_j(0) + g_j(1) = v_{j-1}$ for every $j \in [\mu]$.
(c) $\sum_{i\in[\eta]} \beta^{i-1} \cdot f_i(\boldsymbol{r}_\mu, \zeta'_i) = v_\mu$.

On a full transcript, the first condition states that the verifier of $\mathsf{Prx}$ accepts, by the full-transcript property of $\mathsf{KState}_{\mathsf{Prx}}$, and the other two are the remaining checks of the verifier of $\mathsf{BatchPrx}$. Hence $\mathsf{KState}$ satisfies the full-transcript property.

**Extractor:** The extractor is that of $\mathsf{Prx}$, i.e., $\mathcal{E}_{\text{rbr}}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \rho_j, \mathsf{w}) \coloneqq \mathcal{E}^{\mathsf{Prx}}_{\text{rbr}}(\mathbb{i}', \mathbb{x}', \mathbb{y}', \mathsf{tr}' \,\|\, \rho_j, \mathsf{w})$, where $\mathcal{E}^{\mathsf{Prx}}_{\text{rbr}}$ is the extractor for $\mathsf{Prx}$.

Erasure correction costs $O(\eta \cdot n^3)$ operations, and each invocation of the underlying extractor incurs $O(\eta \cdot n)$ operations to form $\boldsymbol{w}'$, giving the stated total extraction time.

**RBR knowledge soundness.** Let the round-by-round knowledge soundness errors of $\mathsf{Prx}$ be $(\varepsilon_1^{\mathsf{Prx}}, \ldots, \varepsilon_{\mathsf{k}_{\mathsf{Prx}}}^{\mathsf{Prx}})$. We upper bound the round-by-round knowledge soundness errors of $\mathsf{BatchPrx}$ based on the knowledge state function and extractor described above.

1. **Sumcheck batching randomness.** For the first round of $\mathsf{BatchPrx}$, we show that

$$\Pr_{\beta}\left[\exists\, \mathsf{w} : \begin{array}{l} \mathsf{KState}\big(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathcal{E}_{\text{rbr}}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \beta, \mathsf{w})\big) = 0 \\ \wedge\ \mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \beta, \mathsf{w}) = 1 \end{array}\right] \leq \frac{(\eta - 1) \cdot |\Lambda\,(\mathcal{C}^\eta, \delta)\,|}{|\mathbb{F}|} \ .$$

Fix a candidate message tuple in the interleaved decoding list. If an input sumcheck claim is false for this tuple, the difference between the two sides of the batched claim is a nonzero polynomial in $\beta$ of degree at most $\eta - 1$. By Lemma 3.4, the batched claim holds with probability at most $(\eta - 1)/|\mathbb{F}|$. If every input sumcheck claim is true, interleaved list membership implies that each row is $\delta$-close to its corresponding codeword, so the same tuple witnesses the initial relaxed relation. A union bound over the interleaved list gives the claimed error.

2. **Sumcheck rounds.** For each round of $\mathsf{BatchPrx}$ corresponding to round $j \in [\mu]$ of the sumcheck, we show that

$$\Pr_{r_j}\left[\exists\, \mathsf{w} : \begin{array}{l} \mathsf{KState}\big(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathcal{E}_{\text{rbr}}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, r_j, \mathsf{w})\big) = 0 \\ \wedge\ \mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, r_j, \mathsf{w}) = 1 \end{array}\right] \leq \frac{2 \cdot |\Lambda\,(\mathcal{C}^\eta, \delta)\,|}{|\mathbb{F}|} \ .$$

Fix a candidate $\mathsf{w} = [\boldsymbol{m}_i]_{i\in[\eta]}$ such that $[\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}_i)]_{i\in[\eta]} \in \Lambda\left(\mathcal{C}^\eta, [\boldsymbol{w}_i]_{i\in[\eta]}, \delta\right)$ and the deterministic sumcheck checks before round $j$ hold. Define the honest $j$-th partial-sum polynomial

$$h_j(X) \coloneqq \sum_{\boldsymbol{b}'\in\{0,1\}^{\mu-j}} \left( \sum_{i\in[\eta]} \beta^{i-1} \cdot f_i((\boldsymbol{r}_{j-1}, X, \boldsymbol{b}'), \hat{m}_i(\boldsymbol{r}_{j-1}, X, \boldsymbol{b}')) \right) \quad .$$

If $h_j \equiv g_j$, then summing at 0 and 1 and using the verifier's consistency check recovers the preceding sumcheck claim for the same witness. For $j = 1$, this is the batched input claim; for $j > 1$, it is the claim at $\boldsymbol{r}_{j-1}$. An extraction failure therefore requires $h_j \not\equiv g_j$ but $h_j(r_j) = g_j(r_j)$. Both polynomials have degree at most two, since each $f_i$ is multilinear in its arguments and each $\hat{m}_i$ is multilinear. By Lemma 3.4, this event has probability at most $2/|\mathbb{F}|$ for a fixed witness. The claimed error bound follows from taking a union bound over all candidate witnesses in $\Lambda\left(\mathcal{C}^\eta, [\boldsymbol{w}_i]_{i\in[\eta]}, \delta\right)$.

3. **Codeword batching randomness.** For the $(\mu+2)$-th round of BatchPrx, we show that

$$\Pr_{\gamma}\left[\exists\, \mathsf{w} : \begin{array}{l} \mathsf{KState}\big(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathcal{E}_{\text{rbr}}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \gamma, \mathsf{w})\big) = 0 \\ \wedge\ \mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \gamma, \mathsf{w}) = 1 \end{array}\right] \leq \frac{(\eta-1)\cdot|\Lambda\left(\mathcal{C}^\eta, \delta\right)|}{|\mathbb{F}|} + \mathsf{err}^* \quad .$$

The probability that $\gamma$ is not a good challenge for $[\boldsymbol{w}_i]_{i\in[\eta]}$ with respect to Gen is at most $\mathsf{err}^*$ by Definition D.3. It thus suffices to show that the probability that the bad event occurs and $\gamma$ is good is at most $\frac{(\eta-1)\cdot|\Lambda(\mathcal{C}^\eta,\delta)|}{|\mathbb{F}|}$.

Suppose that $\gamma$ is good and that $\mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \gamma, \boldsymbol{m}') = 1$ for some $\boldsymbol{m}'$. Define

$$S \coloneqq \{t \in [n] : \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}')[t] = \boldsymbol{w}'[t]\}.$$

Since $\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}')$ is $\delta$-close to $\boldsymbol{w}'$, the set $S$ has size more than $(1-\delta)n$. As $\delta < \delta_{\mathcal{C}}$, this set is large enough for erasure correction. Furthermore, since $\gamma$ is good and $\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}')[S] = \sum_{i\in[\eta]} \gamma^{i-1}\boldsymbol{w}_i[S]$, there exist (unique) codewords $\boldsymbol{c}_i \in \mathcal{C}$ such that $\boldsymbol{c}_i[S] = \boldsymbol{w}_i[S]$ for every $i \in [\eta]$. Thus the erasure corrector in Extract recovers these codewords and outputs their unique messages $[\boldsymbol{m}_i]_{i\in[\eta]}$, which satisfy $[\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}_i)]_{i\in[\eta]} \in \Lambda\left(\mathcal{C}^\eta, [\boldsymbol{w}_i]_{i\in[\eta]}, \delta\right)$ and, by Lemma D.7,

$$\boldsymbol{m}' = \sum_{i\in[\eta]} \gamma^{i-1}\boldsymbol{m}_i \quad .$$

If moreover $\hat{m}_i(\boldsymbol{r}_\mu) = \zeta_i'$ for every $i \in [\eta]$, then the sumcheck conditions of the pre-challenge knowledge state follow from those of the post-challenge state, and the pre-challenge state is 1 with witness $[\boldsymbol{m}_i]_{i\in[\eta]}$. Hence the bad event occurs only if $\hat{m}_i(\boldsymbol{r}_\mu) \neq \zeta_i'$ for some $i \in [\eta]$, while $\sum_{i\in[\eta]} \gamma^{i-1} \cdot \hat{m}_i(\boldsymbol{r}_\mu) = \hat{m}'(\boldsymbol{r}_\mu) = \zeta' = \sum_{i\in[\eta]} \gamma^{i-1} \cdot \zeta_i'$. For a fixed $[\boldsymbol{m}_i]_{i\in[\eta]}$ with $\hat{m}_i(\boldsymbol{r}_\mu) - \zeta_i' \neq 0$ for some $i$, the polynomial $\sum_{i\in[\eta]} \gamma^{i-1} \cdot \left(\hat{m}_i(\boldsymbol{r}_\mu) - \zeta_i'\right)$ in $\gamma$ is nonzero of degree less than $\eta$, so it vanishes at $\gamma$ with probability at most $(\eta-1)/|\mathbb{F}|$ by Lemma 3.4. The set $\Lambda\left(\mathcal{C}^\eta, [\boldsymbol{w}_i]_{i\in[\eta]}, \delta\right)$ of candidate outputs of Extract does not depend on $\gamma$, so a union bound over it gives the desired bound.

4. **Rounds corresponding to** Prx**.** For each round of BatchPrx corresponding to round $j \in [\mathsf{k}_{\mathsf{Prx}}]$ of Prx, the extractor is that of Prx. The last two conditions of KState do not depend on $\rho_j$, so an extraction failure for BatchPrx is an extraction failure for Prx on the statement $(\mathbb{i}', \mathbb{x}', \mathbb{y}')$ with transcript $\mathsf{tr}' \,\|\, \rho_j$. Hence

$$\Pr_{\rho_j}\left[\exists\, \mathsf{w} : \begin{array}{l} \mathsf{KState}\big(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathcal{E}_{\text{rbr}}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \rho_j, \mathsf{w})\big) = 0 \\ \wedge\ \mathsf{KState}(\mathbb{i}, \mathbb{x}, \mathbb{y}, \mathsf{tr} \,\|\, \rho_j, \mathsf{w}) = 1 \end{array}\right] \leq \varepsilon_j^{\mathsf{Prx}} \quad .$$

This completes the proof of Lemma D.8.

**Partial words and strong monotonicity.** For the partial oracles used by the compiler, undefined columns count as disagreements, and agreement sets contain only fully defined columns. Before invoking the erasure corrector on a partial word, fill its undefined coordinates outside the agreement set $S$ with fixed values. The corrector's output depends only on the coordinates in $S$, so it is independent of this completion. Every bad challenge for partial inputs remains bad for any fixed completion. Since completion can only enlarge the decoding lists, the same error bounds apply. Extending the input oracles preserves list membership and extends $\boldsymbol{w}'$. The plaintext messages $\alpha_i$ are fixed when oracle components are extended, so the deterministic prefix checks are also preserved. Strong monotonicity therefore follows from that of the knowledge states of $\mathsf{Prx}$.

## D.4 Applications to constructing IOPs for NP

We now explain how batching fits into IOPs for NP and how to obtain the IOR used by the Canopy transform of Section 7.2.

**Constructing IOPs for NP.** Many modern IOPs start from a multilinear polynomial IOP (MLIOP), in which the prover sends multilinear polynomial oracles and the verifier queries their evaluations. Compilers such as those in [ACFY25; BMMS26a] replace each polynomial oracle by an encoding of its Boolean evaluation table and reduce the verifier's evaluation queries to SEP claims. If the MLIOP sends $\eta$ polynomial oracles with tables of the same length, the code $\mathcal{C}_0$ is chosen to have that message length. After batching the evaluation claims associated with each oracle, the resulting IOR maps $\mathcal{R}$ to $\mathsf{SEP}[\mathcal{C}_0, \delta_0]^\eta$. Composing it with $\mathsf{BatchPrx}$, under the hypotheses of Lemma D.8, yields an IOP for $\mathcal{R}$.

**Obtaining the IOR of Section 7.2.** The IOPPs of interest are built by composing IORs that successively reduce the message length of the code, followed by a final proximity check. Suppose that the first such IOR, $\mathsf{Reduce}$, maps $\mathsf{SEP}[\mathcal{C}_0, \delta_0]$ to $\mathsf{SEP}[\mathcal{C}, \delta]$. The two batching steps of Appendix D.2 reduce a batch of input SEP claims to a single virtual MEP claim. We view this as a SEP claim using Remark 3.9, then run $\mathsf{Reduce}$ on it. This gives a batched IOR $\mathsf{BatchReduce}$ from $\mathsf{SEP}[\mathcal{C}_0, \delta_0]^\eta$ to $\mathsf{SEP}[\mathcal{C}, \delta]$. The preceding proof supplies its batching extractors and error bounds, while the remaining rounds inherit those of $\mathsf{Reduce}$.

Composing the MLIOP-based IOR with $\mathsf{BatchReduce}$ gives an IOR from $\mathcal{R}$ to $\mathsf{SEP}[\mathcal{C}, \delta]$. Taking $\mathsf{BatchReduce}$ with a single unqueried output oracle gives the setup of Section 7.2, whose remaining proof oracles comprise the $\eta$ purported codewords of $\mathcal{C}_0$ and any intermediate proof oracles.

For the schemes considered here, the verifier receives the code description explicitly, so there is no index oracle. This description is succinct for the Reed–Solomon codes used by WHIR; Bolt permits a verifier whose running time depends on the code description. An IOR with an index oracle can use the generic setup of Section 7.1, committing to that oracle with Merkle trees or Canopy, whose honest commitment binding guarantees are given in Lemmas 4.13 and 6.8.

Finally, the remaining stages of the original IOPP give an IOPP for $\mathsf{SEP}[\mathcal{C}, \delta]$, and hence for $\mathsf{MEP}[\mathcal{C}, \delta]$ by Remark 3.9. This is the base IOPP used to prove the batch of claims produced by the Canopy transform.