Field-Agnostic SNARKs with Small Proofs via Encode-Repeat-Accumulate (ERA) Codes

Anubhav Baweja
abaweja@upenn.edu
UPenn

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Pratyush Mishra
prat@upenn.edu
UPenn

Tushar Mopuri
tmopuri@upenn.edu
UPenn

May 3, 2026

Abstract

Hash-based SNARKs are arguably the most prominent and widely deployed class of transparent SNARKs, owing to desirable properties such as plausible post-quantum security and the avoidance of public-key cryptography. The key ingredients in the construction of hash-based SNARKs are an error-correcting code and an Interactive Oracle Proof of Proximity (IOPP) for that code.

A particularly desirable property for hash-based SNARKs is *field-agnosticism*, i.e., the ability to natively prove statements over arbitrary fields (of sufficient size). We advance the study of field-agnostic hash-based SNARKs as follows:

- We introduce a new family of linear-time encodable field-agnostic error-correcting codes called Encode-Repeat-Accumulate (ERA) codes. These codes achieve a strong trade-off between encoding time and relative distance, the two properties that largely determine the commitment time and proof size of the resulting hash-based SNARK. We also show that ERA codes are efficiently codeswitchable (Ron-Zewi and Rothblum [JACM 2024]), i.e., we can efficiently reduce the task of checking the proximity of a word to an ERA code to checking the proximity of a related word to an arbitrary code.
- We present an IOPP for (interleaved) ERA codes that leverages their fast encoding time, strong distance guarantees, and efficient codeswitchability to achieve concretely small query complexity (and hence small proof size in the resulting hash-based SNARK) while maintaining good prover efficiency.

Contents

1 Introduction **3**
- 1.1 Encode-Repeat-Accumulate codes . . . 4
- 1.2 A query-efficient IOPP for ERA codes . . . 5
- 1.3 Related work . . . 8

2 Technical overview **10**
- 2.1 Distance analysis of ERA codes . . . 10
- 2.2 Query-efficient IOPP for (interleaved) ERA codes . . . 13

3 Distance analysis of ERA codes **16**
- 3.1 Preliminaries . . . 16
- 3.2 Upper bounding $\Pr(\mathcal{F}_{\mathsf{mult1}})$ . . . 17
- 3.3 Upper bounding $\Pr(\mathcal{F}_{\mathsf{mult2}})$ . . . 19
- 3.4 Upper bounding $\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}})$ . . . 22
- 3.5 Computing feasible δ . . . 27

4 IOPP preliminaries **33**
- 4.1 Notation . . . 33
- 4.2 Coding theory . . . 33
- 4.3 Useful results regarding polynomials . . . 35
- 4.4 Relation prerequisites . . . 35
- 4.5 Interactive oracle reductions . . . 35
- 4.6 Important indexed oracle pair relations . . . 37

5 A query-efficient IOPP for interleaved ERA codes **39**
- 5.1 IOPP construction . . . 39
- 5.2 Construction of StartingIOR . . . 40

6 The Reduce IOR **42**
- 6.1 Prerequisites for codeswitching . . . 45
- 6.2 High-level overview of CodeswitchOracles and CodeswitchClaims . . . 47
- 6.3 Full constructions of CodeswitchOracles and CodeswitchClaims . . . 49
- 6.4 Optimizations . . . 55
- 6.5 Concrete efficiency of Reduce . . . 56

7 Implementation and evaluation **58**
- 7.1 The Interleaved IOPP . . . 58
- 7.2 Estimating the performance of our IOPP for ERA codes . . . 60

Acknowledgements **62**

AI use disclosure **62**

References **62**

A Proof of E_{perm} claims **66**
- A.1 Useful facts . . . 66
- A.2 Deferred claims . . . 66

1 Introduction

Hash-based succinct non-interactive arguments of knowledge (SNARKs) are short cryptographic proofs that are quick to verify. They have found widespread applications ranging from efficient blockchains [Eth] to anonymous credentials [FS24], due to their desirable properties, including fast prover and verifier times, plausible post-quantum security, transparent setup, and avoidance of public-key assumptions.

Hash-based SNARKs are constructed by applying the BCS transformation [BCS16] to *Interactive Oracle Proofs* (IOPs) [BCS16; RRR16]: an interactive proof system where the prover can send oracles to the verifier, who can then query these oracles at locations of their choice. Concretely efficient IOPs are constructed using a blueprint that combines two components: (a) a linear code $\mathcal{C}$; and (b) an *IOP of Proximity* (IOPP) for the code $\mathcal{C}$. At a high level, the blueprint works as follows: the IOP prover sends an oracle containing the encoding of the witness under $\mathcal{C}$, and then uses the IOPP to convince the IOP verifier that the oracle is close to a codeword in $\mathcal{C}$ (and, in some settings, that the underlying message satisfies an additional condition). The choice of $\mathcal{C}$ and the IOPP for $\mathcal{C}$ determine two key aspects of the resulting IOP:

1. *The efficiency of the IOP.* The prover time of the IOP depends on the encoding time of $\mathcal{C}$ as well as the prover time of the IOPP, the proof size of the IOP depends on the query complexity of the IOPP, and the verifier time of the IOP depends on both the query complexity and verifier time of the IOPP. Both the query complexity and verifier time of the IOPP are in turn inversely dependent on the distance of $\mathcal{C}$.
2. *The kinds of computations supported.* Certain codes are best suited for fields with special properties, such as FFT-friendly or binary fields, and thus yield IOPs that can only natively support computations defined over such fields. In order to use such IOPs to prove computations defined over other fields, one must first simulate the target arithmetic over the supported field, incurring a significant blow-up in the size of the computation. For example, [BFKTWZ24] note that the circuit that verifies an ECDSA signature on the secp256k1 curve is $\approx 25\times$ smaller when expressed over the native base field of the curve than over an FFT-friendly field.

Because hash-based SNARKs inherit these properties from the underlying IOP, there is significant interest in codes that are simultaneously fast to encode, have good distance, are field-agnostic (i.e., can support arbitrary fields of large enough size), and admit concretely efficient IOPPs. We review the existing landscape through these lenses below, where we use λ to denote the security parameter and N to denote the length of the statements being proved.

Field-specific codes. Reed–Solomon codes have received significant attention in the IOP literature due to being maximum distance separable (MDS) (i.e., offering optimal rate-distance tradeoffs) and their amenability to very efficient IOPP constructions [BBHR18; ACFY24; ACFY25], with the state-of-the-art IOPPs [ACFY24; ACFY25] achieving $O(\lambda \log\log N)$ query complexity and yielding the smallest known concrete proof sizes for hash-based SNARKs. However, Reed–Solomon codes can only be efficiently encoded over FFT-friendly fields, and even then, their encoding time is quasilinear.[1]

Binary Repeat-Accumulate-Accumulate (RAA) codes [DJM98] are linear-time encodable with concretely fast encoding times. In [BCFRRZ25], they were shown to have efficient IOPP constructions [BCFRRZ25] and concretely good distance guarantees, but only over binary fields.

Field-agnostic codes. The Brakedown code [GLSTW23] and Expand-Accumulate codes [BFKTWZ24] are field-agnostic codes with very fast encoding times, but poor distance guarantees. The original works that studied these codes did not present IOPPs tailored to their codes, but instead use the generic 'Interleaved IOPP' construction introduced by Ligero [AHIV17] with $O(\lambda\sqrt{N})$ query complexity, which is prohibitive for

[1] While FFTs for fields underlying elliptic curves can be done in quasilinear time [BCKL22; BCKL23], their practical efficiency is far from competitive [ZCF24].

large N. BrakingBase [NST25] constructs a specialized IOPP for the Brakedown code which has $O(\lambda \log N)$ query complexity, but the concrete efficiency of the IOPP is still impacted by the poor distance guarantees of the Brakedown code.

The BaseFold code [ZCF24] is a field-agnostic code that offers a different tradeoff: it has strong distance guarantees, but it requires significant (quasilinear) time to encode. Additionally, the BaseFold code has a foldable structure that is leveraged in [ZCF24] to construct a tailored IOPP with $O(\lambda \log N)$ query complexity.

In terms of concrete proof sizes, the state-of-the-art field-agnostic hash-based SNARKs [ZCF24; NST25] have proof sizes that are up to $30\times$ larger than those of their field-specific counterparts.

Our contributions. In this paper, we attempt to close the gap between field-agnostic hash-based SNARKs and field-specific hash-based SNARKs via the following contributions:

1. We introduce Encode-Repeat-Accumulate (ERA) codes, a new family of linear-time encodable field-agnostic codes with concretely fast encoding time and strong distance guarantees. For a range of (fast) encoding times, ERA codes achieve better distance guarantees than all prior field-agnostic codes with similar encoding times. We present more details in Section 1.1.

2. We construct a query-efficient IOPP tailored to ERA codes that admits multiple parameter settings with different efficiency trade-offs. For example, it can be parametrized to yield linear prover time, $O(\lambda \log N)$ query complexity and $O(\lambda \log N)$ verifier time. While the asymptotic guarantees of this IOPP are similar to previous work [BCFRRZ25; NST25], the concrete overhead of this IOPP is significantly smaller.

   We compare the concrete performance of our IOPP with previous work by compiling the IOPPs into *multilinear polynomial commitment schemes* (MLPCSs) via Merkle commitments. Our work achieves up to a 60% reduction in proof size compared to state-of-the-art field-agnostic MLPCSs (BaseFold [ZCF24] and BrakingBase [NST25]), while maintaining similar prover times. We present more details in Section 1.2.

1.1 Encode-Repeat-Accumulate codes

An Encode-Repeat-Accumulate code $\mathcal{C} := \mathsf{ERA}(\mathbb{F}, \mathcal{B}, r, \boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{M})$ is a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathsf{ERA}})$[2] where $\mathbb{F}$ is a finite field, $\mathcal{B}$ is a base code with parameters $(\mathbb{F}, n/r, k, \delta_{\mathcal{B}})$, $r \in \mathbb{N}$ is a repetition parameter, $\boldsymbol{P}_1, \boldsymbol{P}_2 \in \{0,1\}^{n\times n} \subseteq \mathbb{F}^{n\times n}$ are permutation matrices, and $\boldsymbol{M} \in \mathbb{F}^{n\times n}$ is a diagonal matrix.

Encoding procedure. A message $\boldsymbol{m} \in \mathbb{F}^k$ is encoded under $\mathcal{C}$ as follows:

1. *Stage 0: encode and repeat*: Encode $\boldsymbol{m} \in \mathbb{F}^k$ under $\mathcal{B}$ to obtain $\boldsymbol{w}_{\mathcal{B}} \leftarrow \mathsf{Enc}_{\mathcal{B}}(\boldsymbol{m}) \in \mathbb{F}^{n/r}$. Then repeat each element of $\boldsymbol{w}_{\mathcal{B}}$ r-many times to obtain $\boldsymbol{w}_0 \in \mathbb{F}^n$.
2. *Stage 1: permute, multiply and accumulate*: Permute the elements of $\boldsymbol{w}_0$ with respect to $\boldsymbol{P}_1$, multiply the resulting vector by $\boldsymbol{M}$, and then apply the accumulate operation to obtain $\boldsymbol{w}_1 \leftarrow \boldsymbol{A} \cdot \boldsymbol{M} \cdot \boldsymbol{P}_1 \cdot \boldsymbol{w}_0$, where the accumulate matrix $\boldsymbol{A} \in \mathbb{F}^{n\times n}$ is the matrix that computes the prefix sum.
3. *Stage 2: permute, multiply and accumulate*: Permute the elements of $\boldsymbol{w}_1$ with respect to $\boldsymbol{P}_2$, multiply the resulting vector by $\boldsymbol{M}$, and then apply the accumulate operation to obtain $\boldsymbol{w}_2 \leftarrow \boldsymbol{A} \cdot \boldsymbol{M} \cdot \boldsymbol{P}_2 \cdot \boldsymbol{w}_1$.

Since the repeat, permute, multiply and accumulate steps are all linear operations, and since the base code $\mathcal{B}$ is linear, $\mathcal{C}$ is a linear code.

While the matrices $\boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{M}$ are of dimension $n \times n$, they can be represented succinctly: a permutation matrix can be represented by a permutation $\pi : [n] \to [n]$, and a diagonal matrix can be represented by its n diagonal entries. The encoding time complexity of $\mathcal{C}$ is $T_{\mathsf{ERA}} = T_{\mathcal{B}} + O(n)$, where $T_{\mathcal{B}}$ is the encoding time

[2] A code with parameters $(\mathbb{F}, n, k, \delta)$ is defined over the field $\mathbb{F}$, and has block length n, message length k, and relative distance δ.

complexity of the base code $\mathcal{B}$. The $O(n)$ term is concretely small, since the repeat, permute, multiply, and accumulate operations together require only $2n$ field additions and $2n$ field multiplications.

Distance guarantees. One can think of ERA codes as 'boosting' the distance of a base code $\mathcal{B}$ that is fast to encode but has poor distance. We show that even for very small values of $\delta_{\mathcal{B}}$, ERA codes have good distance when $\boldsymbol{P}_1, \boldsymbol{P}_2$ are uniformly sampled permutation matrices and $\boldsymbol{M}$ is a uniformly sampled diagonal matrix with *non-zero* diagonal entries. More concretely, we prove the following theorem.

Theorem 1.1 (Informal version of Theorem 3.1)**.** *Let $r \geq 4$ be a repetition parameter, λ be a target failure parameter, and $\mathbb{F}$ be a finite field of size q. Let $\mathcal{B}$ be a linear code with parameters $(\mathbb{F}, n/r, k, \delta_{\mathcal{B}})$. There exists an algorithm* CalcDist[3] *that, on input $(\lambda, r, \delta_{\mathcal{B}}, n, q)$, outputs δ_{ERA} such that the following guarantee holds: if $n = \Omega(\lambda/\delta_{\mathcal{B}})$, and $q = \Omega(\exp(\lambda/(\delta_{\mathcal{B}} n)))$, then with probability at least $1 - 2^{-\lambda}$ over the choice of $\boldsymbol{P}_1$, $\boldsymbol{P}_2$ and $\boldsymbol{M}$, $\mathsf{ERA}(\mathbb{F}, \mathcal{B}, r, \boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{M})$ has relative distance at least δ_{ERA}.*

We present representative outputs of CalcDist in Table 1, which illustrate that the relative distance of ERA codes grows with the repetition parameter r and is not upper bounded by 0.5. This is in contrast to prior work on the concrete distance guarantees of linear-time encodable codes, such as binary Repeat-Accumulate-Accumulate (RAA) codes [BCFRRZ25], Brakedown's code [GLSTW23], as well as concurrent work that analyzes the concrete distance of RAA codes over arbitrary fields [AZ26]. The relative distance guarantees of these works are all upper-bounded by 0.5, stemming from different limitations in their analyses.

base code distance $\delta_{\mathcal{B}}$	repetition factor r								
	4	5	6	8	10	16	25	50	100
0.001	0.2252	0.3212	0.3806	0.4659	0.5292	0.6478	0.7378	0.8351	0.8925
0.003	0.2272	0.3234	0.3882	0.4766	0.5414	0.6611	0.7506	0.8458	0.9011
0.01	0.2293	0.3257	0.3968	0.4906	0.5579	0.6798	0.7686	0.8609	0.9129
0.03	0.2311	0.3278	0.4034	0.5039	0.5746	0.6995	0.7877	0.8764	0.9247

Table 1: Values of δ_{ERA} output by $\mathsf{CalcDist}(\lambda, r, \delta_{\mathcal{B}}, n, q)$ for varying repetition parameters r and base code relative distances $\delta_{\mathcal{B}}$. Here, $\lambda = 100$, $q = 2^{256}$, the message length is always fixed to be 2^{20}, and the base codes all have rate $1/1.1$.

We note that our modular analysis of the distance of ERA codes allows us to reuse the same multiplication matrix across both multiplication steps, which results in better IOPP efficiency, as we discuss in Section 1.2. This is in contrast to the input-output-weight-enumeration (IOWE) based analyses of prior [AR11; AHKM25; HGQZ26] and concurrent [AZ26] work that analyzes the distance of RAA codes over arbitrary fields, all of which require two independently sampled multiplication matrices for the two multiplication steps. We compare our analysis with these works in more detail in Section 1.3.

1.2 A query-efficient IOPP for ERA codes

We provide an overview of our IOPP for ERA codes, which admits several parameter settings with different efficiency trade-offs. We use λ to denote the IOPP soundness parameter, such that the constructed IOPP has soundness error $2^{-\lambda}$, and N to denote the size of the statement being proved.

The Interleaved IOPP. We begin by reviewing what we will denote the 'Interleaved IOPP', a generic IOPP for interleaved linear codes introduced by [AHIV17] (in the context of Reed–Solomon codes) that has

[3] We provide a Python implementation of CalcDist at https://anonymous.4open.science/r/era-code-analysis-1DFF.

subsequently been used in [GLSTW23; BFKTWZ24]. The Interleaved IOPP for the claim "$\boldsymbol{w}$ is close to a codeword in $\mathcal{C}^t$"[4] proceeds in two steps:

1. Fold: proving the claim "$\boldsymbol{w}$ is close to a codeword in $\mathcal{C}^t$" is reduced to proving the claim "$\boldsymbol{w}'$ is close to a codeword in $\mathcal{C}$". The query complexity of this step is $O(-\lambda t/\log(1-\delta_{\mathcal{C}}))$.[5]
2. Base case: the prover sends the message underlying $\boldsymbol{w}'$ to the verifier, which checks that its encoding is close to $\boldsymbol{w}'$.

Because ERA codes have strong distance guarantees, we can directly instantiate the Interleaved IOPP with them and obtain concrete query-complexity improvements over prior instantiations while maintaining similar prover times. For example, we get a 40% reduction in overall proof size of the resulting MLPCS compared to using Brakedown's code [GLSTW23] and Expand-Accumulate codes [BFKTWZ24]. We present this comparison in detail in Section 7.1.

Using codeswitching. A key limitation of the Interleaved IOPP is that in order to balance the costs of the fold and base case steps, the interleaving parameter must be set to $t = \sqrt{N}$. This results in a query complexity of $O(\lambda\sqrt{N})$ for the fold step, which is prohibitive for large statements. Prior work [BCFRRZ25; NST25; BMMS25; BFRW25] has combined the Interleaved IOPP with *codeswitching* [RR24; RR25] to overcome this limitation. Codeswitching is a tool that allows a prover to reduce proving the claim "$\boldsymbol{w}$ is close to a codeword in $\mathcal{C}$" to proving the claim "$\boldsymbol{w}'$ is close to a codeword in $\mathcal{C}'$", for some arbitrary code $\mathcal{C}'$. If this reduction has small overhead in terms of prover time and query complexity, the code $\mathcal{C}$ is said to be *efficiently codeswitchable*.

Our IOPP construction. Our IOPP construction follows the blueprint below, which is obtained by distilling the ideas from [BCFRRZ25; NST25; BMMS25; BFRW25].

Let $\ell \in \mathbb{N}$ be a positive integer, and for each $i \in \{0, 1, \ldots, \ell\}$, let $\mathcal{C}_i$ be a linear code with parameters $(\mathbb{F}, n_i, k_i, \delta_i)$ such that the codes have decreasing message lengths $k_0 > k_1 > \cdots > k_\ell$. Furthermore, we require that for each $i \in \{0, 1, \ldots, \ell - 1\}$, $\mathcal{C}_i$ is efficiently codeswitchable. Let t_0 be a tunable starting interleaving parameter, and for each $i \in [\ell]$, the intermediate interleaving parameter t_i is determined by the choice of $\mathcal{C}_{i-1}$ and $\mathcal{C}_i$ such that $t_i = O(n_{i-1}/k_i)$ (we omit the concrete details of how t_i is defined for brevity, but provide some intuition below). This yields an IOPP for $\mathcal{C}_0^{t_0}$ with the following structure:

1. The first fold: proving the claim "$\boldsymbol{w}_0$ is close to a codeword in $\mathcal{C}_0^{t_0}$" is reduced to proving the claim "$\boldsymbol{w}_0'$ is close to a codeword in $\mathcal{C}_0$". The prover time of this step is $O(n_0 t_0)$ and the query complexity is $O(-\lambda t_0/\log(1-\delta_0))$.
2. For each $i \in [\ell]$:
   (a) Codeswitch: proving the claim "$\boldsymbol{w}_{i-1}'$ is close to a codeword in $\mathcal{C}_{i-1}$" is reduced to proving the claim "$\boldsymbol{w}_i$ is close to a codeword in $\mathcal{C}_i^{t_i}$". The prover time of this step are $O(t_i \cdot T_{\mathcal{C}_i})$, where $T_{\mathcal{C}_i}$ is the encoding time of $\mathcal{C}_i$. At a high-level, the need for $t_i = O(n_{i-1}/k_i)$ arises from codeswitch needing to commit to vectors of total length $O(n_{i-1})$, which it does by breaking them up into t_i many vectors of length k_i and encoding them under $\mathcal{C}_i$. This step can be expressed as a Polynomial IOP (PIOP), and therefore requires no queries.
   (b) Fold: proving the claim "$\boldsymbol{w}_i$ is close to a codeword in $\mathcal{C}_i^{t_i}$" is reduced to proving the claim "$\boldsymbol{w}_i'$ is close to a codeword in $\mathcal{C}_i$". The prover time of this step is $O(n_i t_i)$ and the query complexity is

[4] Given a linear code $\mathcal{C}$ with parameters $(\mathbb{F}, n, k, \delta)$ and an interleaving factor $t \in \mathbb{N}$, the interleaved code $\mathcal{C}^t$ has parameters $(\mathbb{F}^t, n, k, \delta)$ and the codewords of $\mathcal{C}^t$ are obtained by 'stacking' t codewords of $\mathcal{C}$.

[5] We measure query complexity with respect to field elements of $\mathbb{F}$. While it is known how to facilitate a tradeoff between query complexity and alphabet size in the fold step (by performing $O(-\lambda/\log(1-\delta_{\mathcal{C}}))$ 'interleaved queries' over the alphabet $\mathbb{F}^t$ instead of performing $O(-\lambda t/\log(1-\delta_{\mathcal{C}}))$ queries over the alphabet $\mathbb{F}$), the impact of this in our setting is limited. This is because the bottleneck, in terms of proof size of the resulting hash-based MLPCS, is the sheer number of field elements read by the verifier, which is not decreased by performing interleaved queries.

$O(-\lambda t_i / \log(1-\delta_i))$.

3. Base case: The claim "$\boldsymbol{w}'_\ell$ is close to a codeword in $\mathcal{C}_\ell$" is proven with IOPP_ℓ, an IOPP for $\mathcal{C}_\ell$. Let the prover time and query complexity of IOPP_ℓ be $\mathsf{PT}_{\mathsf{IOPP}_\ell}$ and $\mathsf{Q}_{\mathsf{IOPP}_\ell}$ respectively.

We omit the verifier time for brevity, and summarize the overall prover time and query complexity of the blueprint in Table 2. We provide a high-level overview of the codeswitch and fold steps in Section 2.2, and present the full details of our IOPP in Sections 5 and 6.

Prover time	Query complexity
$O\left(n_0 t_0 + \sum_{i=1}^{\ell} t_i(n_i + T_{\mathcal{C}_i}) + \mathsf{PT}_{\mathsf{IOPP}_\ell}\right)$	$O\left(\sum_{i=0}^{\ell} \frac{-\lambda t_i}{\log(1-\delta_i)} + \mathsf{Q}_{\mathsf{IOPP}_\ell}\right)$

Table 2: Prover time and query complexity of the IOPP blueprint.

Codeswitching for ERA codes. Thus, the encoding time and distance of the codes $\mathcal{C}_i$ play a crucial role in determining the efficiency of the IOPP. Since ERA codes combine fast encoding with good distance, they are natural candidates for $\mathcal{C}_i$, provided they are also efficiently codeswitchable. Our main contribution on the IOPP front is a concretely efficient codeswitch reduction for ERA codes when the base code is a tensor code.

Theorem 1.2 (Informal version of Lemma 6.1)**.** *Let $\mathcal{C}_1$ be an* ERA *code with parameters $(\mathbb{F}, n_1, k_1, \delta_1)$ and $\mathcal{C}_2$ be an arbitrary code with parameters $(\mathbb{F}, n_2, k_2, \delta_2)$, such that $k_1 > k_2$ and the base code of $\mathcal{C}_1$ is a tensor code of the form $\mathcal{B} = \mathcal{B}' \otimes \mathcal{B}'$. There exists a codeswitch reduction from a claim of the form "$\boldsymbol{w}_1$ is close to a codeword in $\mathcal{C}_1^{t_1}$" to a claim of the form "$\boldsymbol{w}_2$ is close to a codeword in $\mathcal{C}_2^{t_2}$" with prover time $O(t_1 \cdot n_1 + t_2 \cdot T_{\mathcal{C}_2})$ and query complexity $O(-\lambda t_1 / \log(1-\delta_1))$, where $t_2 < 6n_1/k_2$, $T_{\mathcal{C}_2}$ is the encoding time of $\mathcal{C}_2$, and λ is a target soundness parameter.*

We note that Theorem 1.2 can be generalized to accommodate any base code $\mathcal{B}$ such that the correctness of $\mathcal{B}$'s encoding procedure can be efficiently verified via a Polynomial IOP (PIOP). Prior work gives such PIOPs for tensor codes [BMMS25] as well as Low-Density Parity-Check (LDPC) codes [NST25; BMMS25], the latter of which includes both the Brakedown code and RAA codes. We focus on the case of tensor codes, as the PIOP for tensor codes is more concretely efficient than those for LDPC codes. A concern with using tensor codes is that they have poor distance guarantees: ensuring that $\mathcal{B}$ is fast to encode requires $\mathcal{B}'$ to be fast to encode, resulting in a small $\delta_{\mathcal{B}'}$ and a quadratically smaller $\delta_{\mathcal{B}} = \delta_{\mathcal{B}'}^2$. However, this is not a problem in our setting, since our analysis of ERA codes yields good distance guarantees even for very small values of $\delta_{\mathcal{B}}$.

Minimizing the interleaving factor. As noted in Table 2, the intermediate interleaving factors t_i play a crucial role in determining the efficiency of the IOPP. Accordingly, we put significant effort into minimizing the interleaving factor t in Theorem 1.2. Compared to the naive baseline of $t > 17 \cdot (n_1/k_2)$ that would result from straightforwardly adapting the codeswitch reduction for binary RAA codes in [BCFRRZ25], we are able to reduce t to almost $5 \cdot (n_1/k_2)$. As mentioned earlier, one of the contributing factors towards this improvement is our modular analysis of the distance of ERA codes, which allows us to reuse the same multiplication matrix across both multiplication steps.

Asymptotic efficiency. The claimed asymptotic efficiency of $O(N)$ prover time and $O(\lambda \log N)$ query complexity can be obtained by instantiating the IOPP blueprint with $\ell = \log N$, $t_0 = 1$, setting $\mathcal{C}_0, \mathcal{C}_1, \ldots \mathcal{C}_\ell$ to be a sequence of ERA codes with constant distance and message lengths that decrease by a factor of 2 at each step, and using the same trivial base case as the Interleaved IOPP.

Concrete efficiency. In Section 7.2, we evaluate the concrete performance of the foregoing IOPP blueprint when instantiated with ERA codes, and compare it against prior work on field-agnostic IOPPs [GLSTW23;

ZCF24; BFKTWZ24; NST25]. We use the 256-bit prime field secp256k1 as our underlying field, which is used in the ECDSA signature scheme, as well as on the Bitcoin and Ethereum blockchains, and is not FFT-friendly.

To obtain small proof sizes while maintaining minimal prover time, we set $\ell = 1$. We instantiate $\mathcal{C}_0$ as an ERA code with a small repetition parameter $r = 4$, which keeps encoding fast while still providing good distance, and we instantiate $\mathcal{C}_1$ as the BaseFold code from [ZCF24], a quasilinear-time encodable field-agnostic code with an IOPP of small query complexity. To obtain small proof sizes, we use a BaseFold code with low rate and high distance, relying on the small message length of $\mathcal{C}_1$ to ensure that its concrete encoding time remains small. The hash-based MLPCS obtained from our IOPP achieves up to a 60% reduction in proof size compared to state-of-the-art field-agnostic MLPCSs (BaseFold [ZCF24] and BrakingBase [NST25]), while maintaining similar prover times.

Larger values of ℓ may lead to smaller overall proof sizes in other settings, for example when the underlying field is smaller or when the prover budget is larger. In such regimes, one could leverage the 'Shift to Improve Rate' idea from [ACFY24], which decreases the rate of the intermediate ERA codes as i increases, thereby reducing overall query complexity at the cost of a modest prover-time overhead.

1.3 Related work

We discuss relevant work on concretely efficient IOPPs for codes that are linear-time encodable and/or field-agnostic.

Binary RAA codes. RAA codes over binary fields were first introduced in [DJM98]. [BCFRRZ25] show that they have good distance when defined over $\mathbb{F}_2$, and also introduce Blaze, a specialized IOPP for binary RAA codes that leverages codeswitching. Being defined over $\mathbb{F}_2$, the distance of these codes is upper bounded by 0.5 due to the Plotkin bound [Plo60].

RAA codes over arbitrary fields. [AR11] analyzed RAA codes over arbitrary fields, and showed that as the block length approaches infinity, the relative distance of RAA codes approaches a value close to the Gilbert–Varshamov (GV) bound. However, their analysis heavily relies on these asymptotic assumptions, and does not provide any guarantees for concrete block lengths.

Recently, [AHKM25] analyzed RAA codes over arbitrary large fields, but they do not obtain provable distance guarantees for large block lengths n. Instead, they provide a numerical analysis where they compute failure probabilities for small block lengths, and extrapolate to larger n using regression. Furthermore, their concrete distance guarantees are small (achieving a relative distance of 0.05 for $r = 4$). [HGQZ26] also analyze RAA codes over arbitrary fields, but suffer from similar limitations.

Concurrent work [AZ26] gives concretely strong distance guarantees for RAA codes over arbitrary large fields, for a wide range of block lengths. However, their analysis imposes an upper bound of 0.5 on the relative distance that can be guaranteed. Furthermore, they use the Interleaved IOPP, which also contributes to a large query complexity.

The Brakedown code. This code was first introduced in Brakedown [GLSTW23], building on Spielman's linear-time encodable code [Spi96; DI14], and further analyzed in [Hab23]. It is both linear-time encodable and field-agnostic, but its distance is upper bounded by 0.2 [Hab23] (and is typically smaller than 0.1 for reasonable parameter choices). This results in large proof sizes when used to construct IOPs, and therefore restricts their use to applications where proof size is not a major concern.

BrakingBase [NST25] builds a specialized IOPP for Brakedown's code by building an efficient codeswitching reduction for Brakedown's code. However, the poor distance of Brakedown's code is still a bottleneck for the query complexity of their IOPP.

Expand-Accumulate codes. Expand-Accumulate (EA) codes were introduced in [Boy+22] for $\mathbb{F}_2$ and further analyzed in detail in [BFKTWZ24] for arbitrary large fields. These codes require quasilinear time to encode, but their concrete encoding times are quite fast. They achieve provable distance guarantees similar to Brakedown's code. To the best of our knowledge, there do not exist any specialized IOPPs for EA codes.

BaseFold. [ZCF24] introduce the BaseFold code, a quasilinear-time encodable field-agnostic code with strong distance guarantees. Unlike EA codes, the concrete encoding time of the BaseFold code is considerable, but [ZCF24] present a specialized IOPP for the BaseFold code that achieves competitive query complexity.

Bolt and Lightning. Concurrent works [GNR26; QGZ26] introduce *sketched codes*, linear codes that are heavily optimized for prover time. The relative distance of these codes ranges in the 0.001 to 0.01 range, which leads to IOPPs with large query complexity.

2 Technical overview

In this section, we provide a high-level overview of our distance analysis of ERA codes, the construction of an IOPP for ERA codes, as well as some directions for future work.

2.1 Distance analysis of ERA codes

In Section 3, we show that with high probability over uniformly sampled permutation matrices $\boldsymbol{P}_1, \boldsymbol{P}_2$ and non-zero diagonal matrix $\boldsymbol{M}$, the code $\mathsf{ERA}(\mathbb{F}, \mathcal{B}, r, \boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{M})$ has good relative distance. In this section, we give a high-level overview of the technical ideas underlying our distance analysis. We do so by analyzing the distance of the *single-stage* ERA code $\mathcal{C}' := \mathsf{ERA}(\mathbb{F}, \mathcal{B}, r, \boldsymbol{P}, \boldsymbol{M})$, wherein we skip the second stage of the encoding procedure, as such an analysis captures our main ideas.

We first analyze the behavior of the accumulate step, via the notion of intervals. This notion has been used in the analysis of Repeat-Accumulate-Accumulate (RAA) codes [DJM98; AR11; BCFRRZ25; AHKM25; AZ26] and Expand-Accumulate (EA) codes [BFKTWZ24], both of which utilize accumulate steps.

Definition 2.1 (informal (Definitions 3.3 and 3.4))**.** Consider $\boldsymbol{w} \in \mathbb{F}^n$ such that $\mathsf{wt}(\boldsymbol{w}) = w$[6], containing non-zero elements in indices $\alpha_1 < \ldots < \alpha_w$. The *$j$-th interval* of $\boldsymbol{w}$ is defined as the set of indices $I_j = [\alpha_j, \ldots, \alpha_{j+1} - 1]$ (with I_0 being the indices before α_1, and I_w being the indices after α_w).

For any interval I_j, the vector obtained by applying a prefix sum to $\boldsymbol{w}$, $\boldsymbol{a} := \boldsymbol{A} \cdot \boldsymbol{w}$, contains the same element in every index $i \in I_j$. We say I_j is an *on-interval* if $\boldsymbol{a}$ has non-zero elements in I_j; otherwise, we say that I_j is an *off-interval*.

Prior analyses of the accumulate step. Many prior works that analyze the concrete distance of codes with accumulate steps (e.g., RAA codes [BCFRRZ25] and EA codes [BFKTWZ24]) have used the following guarantee: for any vector $\boldsymbol{w} \in \mathbb{F}_2^n$ with $\mathsf{wt}(\boldsymbol{w}) = w$ that is to be accumulated, every off-interval I_j is followed by an on-interval I_{j+1}, implying that at least half of the intervals are on-intervals. It is then shown, using either a permute step or an expand step, that with high probability the sum of the lengths of these $\lceil w/2 \rceil$ on-intervals of $\boldsymbol{w}$ is large, implying that the weight of the accumulated vector $\boldsymbol{A} \cdot \boldsymbol{w}$ is large.

The foregoing idea is tight over $\mathbb{F}_2$, where an on-interval must be followed by an off-interval since the only non-zero element $1 \in \mathbb{F}_2$ always flips the parity of the accumulated sum. However, over larger fields, an on-interval may be followed by another on-interval.

Our analysis. We take advantage of the foregoing possibility by introducing the *multiply* step; we show that with high probability over $\boldsymbol{M}$, nearly $(1 - 1/r)$-fraction of intervals of the vector obtained after the multiply step are on-intervals. We provide a more detailed overview of our analysis by sketching a proof of the following informal lemma, which is a simplified version of Theorem 3.1, our main theorem for the analysis of the two-stage ERA code.

Lemma 2.2 (informal)**.** *There exists an algorithm* $\mathsf{CalcDistSimple}(\lambda, r, \delta_{\mathcal{B}})$ *that outputs* δ_{ERA}*, such that for any base code* $\mathcal{B}$ *with parameters* $(\mathbb{F}, n/r, k, \delta_{\mathcal{B}})$*,* $\mathcal{C}'$ *has relative distance at least* δ_{ERA} *with probability at least* $\left(1 - 2^{-\lambda}\right)$ *if* n *and* $|\mathbb{F}|$ *are sufficiently large.*

We will work our way up to the construction of CalcDistSimple. Let $\delta_{\mathsf{ERA}} \leftarrow \mathsf{CalcDistSimple}(\lambda, r, \delta_{\mathcal{B}})$. Since $\mathcal{C}'$ is a linear code, it suffices to show that for any non-zero message $\boldsymbol{m} \in \mathbb{F}^k$, $\mathsf{wt}(\mathsf{Enc}_{\mathcal{C}'}(\boldsymbol{m})) \geq \delta_{\mathsf{ERA}} n$. We use $\boldsymbol{R} \in \mathbb{F}^{n \times n/r}$ and $\boldsymbol{A} \in \mathbb{F}^{n \times n}$ to denote the matrices that implement the repeat and accumulate operations respectively. For brevity, while considering a particular input message $\boldsymbol{m} \in \mathbb{F}^k$, we often directly

[6] We use $\mathsf{wt}(\boldsymbol{v})$ to denote the Hamming weight of the vector $\boldsymbol{v}$.

denote $\boldsymbol{w}_0 := \boldsymbol{R} \cdot \boldsymbol{G}_{\mathcal{B}} \cdot \boldsymbol{m}$, $\boldsymbol{w}_1 := \boldsymbol{M} \cdot \boldsymbol{P} \cdot \boldsymbol{w}_0$, and $\boldsymbol{w}_2 := \boldsymbol{A} \cdot \boldsymbol{w}_1$. Let $c > 1$ be a fixed constant. We define the following failure events in terms of r, c, δ:

1. *Event* $\mathcal{F}_{\mathsf{mult}}$*:* There exists a non-zero message $\boldsymbol{m} \in \mathbb{F}^k$ such that the number of on-intervals in $\boldsymbol{w}_1$ is less than $\mathsf{wt}(\boldsymbol{w}_1) \cdot (1 - c/r)$.

2. *Event* $\mathcal{F}_{\mathsf{perm}}(\delta)$*:* There exists a non-zero message $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_2) < \delta n$.

Clearly, if $\mathcal{F}_{\mathsf{perm}}(\delta)$ does not occur, then the code $\mathcal{C}'$ has relative distance $\geq \delta$. Furthermore, we have that

$$\Pr(\mathcal{F}_{\mathsf{perm}}(\delta)) \leq \Pr(\mathcal{F}_{\mathsf{mult}}) + \Pr(\mathcal{F}_{\mathsf{perm}}(\delta) \mid \neg\mathcal{F}_{\mathsf{mult}}) \ .$$

Therefore, in the rest of this section we will focus on showing that $\Pr(\mathcal{F}_{\mathsf{mult}}) \leq 2^{-\lambda-1}$ and $\Pr(\mathcal{F}_{\mathsf{perm}}(\delta_{\mathsf{ERA}}) \mid \neg\mathcal{F}_{\mathsf{mult}}) \leq 2^{-\lambda-1}$.

Upper bounding $\Pr(\mathcal{F}_{\mathsf{mult}})$. Define $\mathcal{F}_{\mathsf{mult}}(\ell)$ as the event that there exists a non-zero $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = \ell$ and the number of on-intervals in $\boldsymbol{w}_1$ is $< \ell(1 - c/r)$.

Claim 2.3 (informal)**.** For any fixed $\ell \in [n]$ and constant $c > 1$, we have

$$\Pr(\mathcal{F}_{\mathsf{mult}}(\ell)) \leq \binom{n/r}{\ell/r} \cdot (q-1)^{\ell/r} \cdot \left(\frac{er}{c(q-1)}\right)^{c\ell/r} \ .$$

Proof sketch. The number of non-zero messages $\boldsymbol{m} \in \mathbb{F}^k$ such that the weight of the corresponding $\boldsymbol{w}_0$ is ℓ is at most the number of such intermediate words $\boldsymbol{w}_0$ themselves, which is $\leq \binom{n/r}{\ell/r} \cdot (q-1)^{\ell/r}$. This is because any $\boldsymbol{w}_0$ must be obtained by repeating some $\boldsymbol{w}_{\mathcal{B}} := \boldsymbol{G}_{\mathcal{B}} \cdot \boldsymbol{m}$ of weight ℓ/r, so there are $\binom{n/r}{\ell/r}$ choices for the support and $(q-1)^{\ell/r}$ choices for the non-zero values.

Let $\mathcal{F}_{\mathsf{mult}}(\ell, \boldsymbol{m})$ be the event that for a fixed non-zero message $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = \ell$, the number of on-intervals in $\boldsymbol{w}_1$ is $< \ell(1 - c/r)$. Clearly, $\boldsymbol{w}_1$ is a vector with weight ℓ such that all its non-zero elements are uniformly random and independent. Let the indices of non-zero elements of $\boldsymbol{w}_1$ be $\alpha_1 < \alpha_2 < \ldots < \alpha_\ell$.

Let X_j be a Bernoulli random variable that is 1 if and only if the j-th interval I_j of $\boldsymbol{w}_1$ is an off-interval. For each $j \in [\ell]$, the j-th interval I_j of $\boldsymbol{w}_1$ is an off-interval if and only if $\boldsymbol{w}_1[\alpha_j] = -\sum_{i=1}^{j-1} \boldsymbol{w}_1[\alpha_i]$. Therefore, for all $j \geq 2$, $\Pr[X_j = 1] = 1/(q-1)$. Defining $X := \sum_{j=2}^{\ell} X_j$ as the number of off-intervals in $\boldsymbol{w}_1$, we can use the Chernoff bound[7] to show that

$$\mathcal{F}_{\mathsf{mult}}(\ell, \boldsymbol{m}) = \Pr[X > c\ell/r] \leq \left(\frac{e^{(c(q-1)/r)-1}}{(c(q-1)/r)^{c(q-1)/r}}\right)^{\ell/(q-1)} \leq \left(\frac{er}{c(q-1)}\right)^{c\ell/r} \ .$$

Applying the union bound over all $\boldsymbol{m}$ completes the proof. □

Now we utilize the *encode* step, which guarantees that for all $0 \leq \ell < \lceil \delta_{\mathcal{B}} n \rceil$, there are no non-zero messages $\boldsymbol{m}$ such that $\mathsf{wt}(\boldsymbol{w}_0) = \ell$, and therefore $\Pr(\mathcal{F}_{\mathsf{mult}}(\ell)) = 0$ for all such ℓ. Thus, we have

$$\Pr(\mathcal{F}_{\mathsf{mult}}) \leq \sum_{\ell=\lceil \delta_{\mathcal{B}} n \rceil}^{n} \Pr(\mathcal{F}_{\mathsf{mult}}(\ell)) \leq \sum_{\ell=\lceil \delta_{\mathcal{B}} n \rceil}^{n} \binom{n/r}{\ell/r} \cdot (q-1)^{\ell/r} \cdot \left(\frac{er}{c(q-1)}\right)^{c\ell/r}$$

[7] Since X_j are not independent, using the Chernoff bound is not strictly justified. In the formal proof, we define new Bernoulli random variables that stochastically dominate X_j and are independent.

$$\leq \sum_{\ell=\lceil \delta_{\mathcal{B}} n \rceil}^{n} \left(\frac{en}{\ell}\right)^{\ell/r} \cdot \left(\left(\frac{er}{c}\right)^{c} \cdot (q-1)^{1-c}\right)^{\ell/r} \leq n \cdot \left(\frac{e}{\delta_{\mathcal{B}}} \cdot \left(\frac{er}{c}\right)^{c} \cdot (q-1)^{1-c}\right)^{\ell/r} .$$

Rearranging, we have $\Pr(\mathcal{F}_{\mathsf{mult}}) \leq 2^{-\lambda-1}$ if

$$q - 1 \geq \left(\frac{e}{\delta_{\mathcal{B}}} \cdot \left(\frac{er}{c}\right)^{c} \cdot \exp_2\left(\frac{r(\lambda + 1 + \log n)}{\delta_{\mathcal{B}} n}\right)\right)^{\frac{1}{c-1}} .$$

Upper bounding $\Pr(\mathcal{F}_{\mathsf{perm}}(\delta) \mid \neg\mathcal{F}_{\mathsf{mult}})$. Similar to the analysis of $\Pr(\mathcal{F}_{\mathsf{mult}})$, define $\mathcal{F}_{\mathsf{perm}}(\ell, \delta)$ as the event that there exists a non-zero $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = \ell$ and $\mathsf{wt}(\boldsymbol{w}_2) < \delta n$. In Claim 2.3, we were able to upper bound $\Pr(\mathcal{F}_{\mathsf{mult}}(\ell))$ by applying the union bound over more than $q^{\ell/r}$ input messages $\boldsymbol{m}$. However, this strategy will not suffice here. This probability is analyzed over $\boldsymbol{P}$, but there are only $n!$ many such permutations, which is much smaller than $q^{\ell/r}$ if q is large, precluding such a union bound. However, we can apply a union bound over *configurations* instead.

Definition 2.4 (Configurations)**.** Given a vector $\boldsymbol{w} \in \mathbb{F}^n$, we define its *configuration* $\overline{\boldsymbol{w}} \in \{0,1\}^n$ as the vector that has 1 in the positions where $\boldsymbol{w}$ has non-zero elements, and 0 elsewhere. We use $C_{n,\ell}$ to denote the set of all configurations of length n and weight ℓ.

The key observation is that if there are two messages $\boldsymbol{m}, \boldsymbol{m}' \in \mathbb{F}^k$ such that $\boldsymbol{MPRG}_{\mathcal{B}}\boldsymbol{m}$ and $\boldsymbol{MPRG}_{\mathcal{B}}\boldsymbol{m}'$ have the same configuration, then their respective interval lengths are also the same. If we can show that for a given configuration, the probability that there exists a subset of $m > \ell(1 - c/r)$ intervals whose lengths sum to $< \delta n$ is small, then we can apply a union bound over all configurations in $C_{n,\ell}$ to upper bound $\Pr(\mathcal{F}_{\mathsf{perm}}(\ell, \delta) \mid \neg\mathcal{F}_{\mathsf{mult}})$. The number of such configurations is only $\binom{n/r}{\ell/r}$ (independent of q), so applying a union bound over all configurations is feasible. This motivates us to focus on upper bounding the following probability for a fixed configuration $\overline{\boldsymbol{w}}_1 \in C_{n,\ell}$:

Claim 2.5 (informal)**.** Let $\mathcal{F}_{\mathsf{perm}}(\ell, \overline{\boldsymbol{w}}_1, \delta)$ be the event that for a fixed configuration $\overline{\boldsymbol{w}}_1 \in C_{n,\ell}$ (the configuration of $\boldsymbol{w}_1 := \boldsymbol{MPRG}_{\mathcal{B}}\boldsymbol{m}$), the weight of $\boldsymbol{w}_2$ is $< \delta n$. There exists a function $f(\ell, \delta)$ such that

$$\Pr(\mathcal{F}_{\mathsf{perm}}(\ell, \overline{\boldsymbol{w}}_1, \delta) \mid \neg\mathcal{F}_{\mathsf{mult}}) \leq \mathrm{poly}(n) \cdot 2^{(n \cdot f(\ell,\delta))} .$$

The proof of Claim 2.5 and the definition of $f(\ell, \delta)$ are quite involved, so we refer the reader to Section 3.4 for the detailed proof of the analogous two-stage claim (specifically, Lemma 3.16). Applying the union bound over choices of ℓ and $\overline{\boldsymbol{w}}_1$, we get

$$\Pr(\mathcal{F}_{\mathsf{perm}}(\delta) \mid \neg\mathcal{F}_{\mathsf{mult}}) \leq \sum_{\ell=\lceil \delta_{\mathcal{B}} n \rceil}^{n} \binom{n/r}{\ell/r} \cdot \Pr(\mathcal{F}_{\mathsf{perm}}(\ell, \overline{\boldsymbol{w}}_1, \delta) \mid \neg\mathcal{F}_{\mathsf{mult}}) \leq \sum_{\ell=\lceil \delta_{\mathcal{B}} n \rceil}^{n} \mathrm{poly}(n) \cdot 2^{(n \cdot f'(\ell,\delta))} ,$$

where f' is a slight modification of f that incorporates the $\binom{n/r}{\ell/r}$ term. The summand $f'(\ell, \delta)$ is maximized at $\ell = \lceil \delta_{\mathcal{B}} n \rceil$, since larger values of ℓ correspond to more non-zero elements in $\boldsymbol{w}_0$, which should only help increase the weight of $\boldsymbol{w}_2$.[8] Therefore, we get

$$\Pr(\mathcal{F}_{\mathsf{perm}}(\delta) \mid \neg\mathcal{F}_{\mathsf{mult}}) \leq \mathrm{poly}(n) \cdot 2^{(n \cdot f'(\lceil \delta_{\mathcal{B}} n \rceil,\delta))} .$$

[8] While this is true for the single-stage analysis, it is surprisingly not true for the two-stage analysis discussed in Section 3.4. This further complicates the analysis and computation of the optimal δ_{ERA}, and is detailed in Section 3.5.

Computing δ_{ERA}. We are now just left to choose δ_{ERA}, the largest value of δ that still ensures that $\mathrm{poly}(n) \cdot 2^{(n \cdot f'(\lceil \delta_{\mathcal{B}} n \rceil, \delta))} \leq 2^{-\lambda - 1}$. f' is a transcendental function in δ, so we have to use numerical methods to compute such a value. $f'(\lceil \delta_{\mathcal{B}} n \rceil, \delta)$ is monotonically increasing in δ, so we can perform binary search over $\delta \in (0, 1)$ to find the largest δ such that $f'(\lceil \delta_{\mathcal{B}} n \rceil, \delta) \leq -\gamma(n, \lambda)$ for some appropriate $\gamma(n, \lambda) > 0$ such that $\mathrm{poly}(n) \cdot 2^{-\gamma(n,\lambda) \cdot n} \leq 2^{-\lambda - 1}$. This process defines the algorithm $\mathsf{CalcDistSimple}(\lambda, r, \delta_{\mathcal{B}})$ mentioned in Lemma 2.2, completing the proof sketch.

2.2 Query-efficient IOPP for (interleaved) ERA codes

The IOPP blueprint in Section 1.2 involves the sequential application of two separate codeswitching and folding reductions. While it is conceptually simpler to use separate reductions for these steps, in our actual IOPP construction, we roll these two reductions into a single concretely efficient reduction that we call Reduce. We formalize Reduce using the language of Interactive Oracle Reductions (IORs) [BCGGRS19; BMNW25], which we recall below.

IORs. An *oracle relation* $\mathcal{R}$ is a set of triples of the form $(\mathbb{x}, \vec{\mathbb{y}}, \mathbb{w})$, where $\mathbb{x}$ is the explicit instance, $\vec{\mathbb{y}}$ is an implicit instance specified as a set of vectors that the verifier has oracle access to, and $\mathbb{w}$ is the witness. An IOR from an oracle relation $\mathcal{R}$ to an oracle relation $\mathcal{R}'$ is an interactive protocol that allows a prover $\mathbf{P}$ to reduce the task of convincing a verifier $\mathbf{V}$ that an input instance $(\mathbb{x}, \vec{\mathbb{y}})$ is in the language $\mathcal{L}(\mathcal{R})$ to convincing $\mathbf{V}$ that the output instance $(\mathbb{x}', \vec{\mathbb{y}}')$ is in $\mathcal{L}(\mathcal{R}')$. IORs are discussed in detail in Section 4.5.

IOPPs for arbitrary codes. A detail that we glossed over in Section 1 is that in order to use an arbitrary code $\mathcal{C}$ (i.e. one that is not a Reed–Solomon or Reed–Muller code) to construct an IOP, the IOPP for $\mathcal{C}$ must prove not only that an oracle is close to a codeword of $\mathcal{C}$, but also that the underlying message satisfies an *evaluation claim*. This is sometimes referred to more explicitly as an IOPP for multilinear evaluation [BCFRRZ25]. This notion was formalized in the language of IORs in [BMMS25] via the following oracle relation.

Definition 2.6 (informal)**.** Let $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$, and let $\varepsilon \in (0, \delta_{\mathcal{C}})$. The *multilinear evaluation* oracle relation $\mathsf{ME}^{(\mathcal{C}, \varepsilon)}$ consists of tuples of the form

$$(\mathbb{x} = (\boldsymbol{z}, y), \vec{\mathbb{y}} = \boldsymbol{w}, \mathbb{w} = \boldsymbol{m}) \ ,$$

such that $\delta(\boldsymbol{w}, \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m})) \leq \varepsilon$, which we henceforth denote by $\boldsymbol{w} \sim_{\varepsilon} \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m})$, and for the multilinear extension $\hat{m}$ of $\boldsymbol{m}$, we have $\hat{m}(\boldsymbol{z}) = y$.

An IOPP for $\mathcal{C}$ is an IOR from $\mathsf{ME}^{(\mathcal{C}, \varepsilon)}$ to the trivial relation $\mathcal{R}_{\mathsf{true}} := \{(\mathbb{x} := \mathsf{accept}, \vec{\mathbb{y}} := \bot, \mathbb{w} := \bot)\}$. Accordingly, Reduce reduction must also involve such evaluation claims.

The Reduce IOR. We formalize the key reduction in our IOPP construction as Reduce, an IOR from $\mathsf{ME}^{(\mathsf{ERA}, \varepsilon)}$ to $\mathsf{ME}^{(\mathcal{C}', \varepsilon')}$, where ERA is an ERA code with parameters $(\mathbb{F}, n, k, \delta_{\mathsf{ERA}})$ and $\mathcal{C}'$ is an arbitrary linear code with parameters $(\mathbb{F}, n', k', \delta_{\mathcal{C}'})$ such that $k' < k$. An important prerequisite for the construction of the Reduce IOR is the following *inner product* oracle relation.

Definition 2.7 (informal)**.** Let $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$, and let $\varepsilon \in (0, \delta_{\mathcal{C}})$. The *inner product* oracle relation $\mathsf{IP}^{(\mathcal{C}, \varepsilon)}$ consists of tuples of the form

$$(\mathbb{x} = (\boldsymbol{v}, \sigma), \vec{\mathbb{y}} = \boldsymbol{w}, \mathbb{w} = \boldsymbol{m}) \ ,$$

such that $\boldsymbol{w} \sim_{\varepsilon} \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m})$, and $\langle \boldsymbol{v}, \boldsymbol{m} \rangle = \sigma$.

The $\mathsf{IP}^{(\mathcal{C},\varepsilon)}$ relation is a special case of the sumcheck evaluation relation from [BMMS25], which would phrase the inner product claim as a sumcheck claim $\sum_{\boldsymbol{b}\in\{0,1\}^{\log k}} \hat{m}(\boldsymbol{b}) \cdot \hat{v}(\boldsymbol{b}) = \sigma$, and allows for more general sumcheck claims as well. This relation allows the prover and verifier to set up a very expressive inner product claim, and then use the sumcheck protocol on the foregoing sumcheck interpretation of the claim to reduce it to checking a much simpler multilinear evaluation claim.

We are now ready to give a high-level description of the Reduce IOR, which proceeds in three steps.

First step: breaking up the evaluation claim. We assume WLOG that k is a multiple of k', and define $\ell_{\boldsymbol{m}} := k/k'$. The prover $\mathbf{P}$ holds a witness $\boldsymbol{m} \in \mathbb{F}^k$ and wants to convince the verifier $\mathbf{V}$ that

$$\boldsymbol{w} \sim_\varepsilon \mathsf{Enc}_{\mathsf{ERA}}(\boldsymbol{m}) \ , \quad \text{and} \quad \hat{m}(\boldsymbol{z}) = y \ .$$

$\mathbf{P}$ first splits its message $\boldsymbol{m}$ into $\ell_{\boldsymbol{m}}$ blocks of size k', denoted by $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{\ell_{\boldsymbol{m}}} \in \mathbb{F}^{k'}$. $\mathbf{P}$ and $\mathbf{V}$ parse $\boldsymbol{z} = (\boldsymbol{z}_1, \boldsymbol{z}_2)$, where $\boldsymbol{z}_1 \in \mathbb{F}^{\log k - \log k'}$ and $\boldsymbol{z}_2 \in \mathbb{F}^{\log k'}$. Then for each $i \in [\ell_{\boldsymbol{m}}]$:

1. $\mathbf{P}$ commits to $\boldsymbol{m}_i$ by computing $\boldsymbol{w}_i := \mathsf{Enc}_{\mathcal{C}'}(\boldsymbol{m}_i) \in \mathbb{F}^{n'}$ and sending the oracle $[\![\boldsymbol{w}_i]\!]$ to $\mathbf{V}$.
2. $\mathbf{P}$ sends $y_i := \hat{m}_i(\boldsymbol{z}_2)$ to $\mathbf{V}$.

The lagrange basis polynomial $\mathrm{eq}_\mu(\boldsymbol{X}, \boldsymbol{Y}) := \prod_{i=1}^{\mu}(X_i Y_i + (1 - X_i)(1 - Y_i))$, can be used to express multilinear evaluation claims as inner product claims (we omit denoting μ henceforth since it will be clear from context). In particular, using $\langle i \rangle$ to represent the binary representation of i, we have

$$\hat{m}(\boldsymbol{z}) = y \quad \iff \quad \sum_{i=1}^{\ell_{\boldsymbol{m}}} \mathrm{eq}(\boldsymbol{z}_1, \langle i-1 \rangle) \cdot \hat{m}_i(\boldsymbol{z}_2) = y \ .$$

$\mathbf{V}$ then checks that the claimed evaluations $[y_i]_{i\in[\ell_{\boldsymbol{m}}]}$ satisfy $\sum_{i=1}^{\ell_{\boldsymbol{m}}} \mathrm{eq}(\boldsymbol{z}_1, \langle i-1 \rangle) \cdot y_i = y$.

Finally, $\mathbf{V}$ wants to check that for each $i \in [\ell_{\boldsymbol{m}}]$, it is indeed the case that $\hat{m}_i(\boldsymbol{z}_2) = y_i$ where $\boldsymbol{m}_i$ is the message underlying the oracle $[\![\boldsymbol{w}_i]\!]$. Defining $\mathbf{eq}(\boldsymbol{z}_2) := [\mathrm{eq}(\boldsymbol{z}_2, \boldsymbol{b})]_{\boldsymbol{b}\in\{0,1\}^{\log k'}}$, this can be expressed as the inner product claims in $\mathsf{IP}^{(\mathcal{C}',\varepsilon')}$ claims of the form

$$\forall i \in [\ell_{\boldsymbol{m}}] : ((\mathbf{eq}(\boldsymbol{z}_2), y_i), [\![\boldsymbol{w}_i]\!], \boldsymbol{m}_i) \in_? \mathsf{IP}^{(\mathcal{C}',\varepsilon')} \ .$$

This convinces $\mathbf{V}$ that for the $\boldsymbol{m}^* := (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{\ell_{\boldsymbol{m}}})$ underlying the $[\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_{\ell_{\boldsymbol{m}}}]\!]$, we have $\hat{m}^*(\boldsymbol{z}) = y$.

Second step: spot checks. $\mathbf{P}$ must still convince $\mathbf{V}$ that this message $\boldsymbol{m}^*$ is consistent with the original message whose encoding is close to $[\![\boldsymbol{w}]\!]$. This is done via codeswitching techniques developed in [RR24; RR25; BCFRRZ25; BMMS25], where $\mathbf{V}$ first samples $\nu = O(-\lambda/\log(1-\varepsilon))$ random indices $\alpha_1, \ldots, \alpha_\nu \xleftarrow{\$} [n]$, and queries $[\![\boldsymbol{w}]\!]$ at these indices to obtain $\sigma_1, \ldots, \sigma_\nu$. $\mathbf{P}$ now convinces $\mathbf{V}$ that the following 'spot checks' hold:

$$\forall j \in [\nu] : \sigma_j = \mathsf{Enc}_{\mathsf{ERA}}(\boldsymbol{m}^*)[\alpha_j] \ .$$

We briefly describe a few details of how this is done:

- $\mathbf{P}$ commits to the codeword $\boldsymbol{w}^{\mathsf{ERA}} := \mathsf{Enc}_{\mathsf{ERA}}(\boldsymbol{m}^*)$ by splitting it into $\ell_{\mathsf{ERA}} := n/k'$ blocks of size k', denoted by $\boldsymbol{w}_1^{\mathsf{ERA}}, \ldots, \boldsymbol{w}_{\ell_{\mathsf{ERA}}}^{\mathsf{ERA}} \in \mathbb{F}^{k'}$, computing $[\boldsymbol{o}_i^{\mathsf{ERA}}]_{i\in[\ell_{\mathsf{ERA}}]}$ where $\boldsymbol{o}_i^{\mathsf{ERA}} := \mathsf{Enc}_{\mathcal{C}'}(\boldsymbol{w}_i^{\mathsf{ERA}})$ for each $i \in [\ell_{\mathsf{ERA}}]$, and sending the oracles $[\![\boldsymbol{o}_1^{\mathsf{ERA}}]\!], \ldots, [\![\boldsymbol{o}_{\ell_{\mathsf{ERA}}}^{\mathsf{ERA}}]\!]$ to $\mathbf{V}$.
- $\mathbf{P}$ convinces $\mathbf{V}$ that for the $\boldsymbol{w}^{\mathsf{ERA}}$ underlying $[\![\boldsymbol{o}_1^{\mathsf{ERA}}]\!], \ldots, [\![\boldsymbol{o}_{\ell_{\mathsf{ERA}}}^{\mathsf{ERA}}]\!]$, we have $\boldsymbol{w}^{\mathsf{ERA}}[j] = \sigma_j$ for each $j \in [\nu]$. It does so by setting up appropriate $\mathsf{IP}^{(\mathcal{C}',\varepsilon')}$ inner product claims with respect to the oracles $[\![\boldsymbol{o}_1^{\mathsf{ERA}}]\!], \ldots, [\![\boldsymbol{o}_{\ell_{\mathsf{ERA}}}^{\mathsf{ERA}}]\!]$.
- $\mathbf{P}$ is just left to show that the committed $\boldsymbol{w}^{\mathsf{ERA}}$ is obtained by applying the encoding procedure of ERA to $\boldsymbol{m}^*$, which is done as follows:

– To prove that the encode step was performed correctly, $\mathbf{P}$ commits to $\boldsymbol{w}^{\mathcal{B}} \in \mathbb{F}^{n_{\mathcal{B}}}$ (again, by splitting it up into blocks and encoding them under $\mathcal{C}'$) and uses the tensor codeswitching technique from [BMMS25] to convince $\mathbf{V}$ that $\boldsymbol{w}^{\mathcal{B}}$ is the result of encoding $\boldsymbol{m}^*$ under the base code $\mathcal{B}$ of ERA. This requires us to set the base code $\mathcal{B}$ to be a tensor code of the form $\mathcal{B} = \mathcal{B}' \otimes \mathcal{B}'$, for some linear code $\mathcal{B}'$.
– We must also prove that the repeat, permute, multiply, and accumulate steps were performed correctly. To do so, we build on the codeswitch for RAA codes from [BCFRRZ25], which at a high level involve committing to the sequence of intermediate vectors obtained after applying each of these steps, and then proving that each step was performed correctly using various inner product claims. Naively adapting their approach to ERA codes would involve committing to a large number of intermediate vectors, which amounts (looking ahead) to increasing the interleaving factor t of the output interleaved code. We show that we can avoid committing to several of these intermediate vectors, and instead 'simulate' them using inner product claims. Furthermore, the permutation check used in [BCFRRZ25] would require committing to 4 additional permutation witness vectors of length $2\ell_{\mathsf{ERA}}$ each. We replace this permutation check with the one from [PH23], which does not require committing to any additional permutation witness vectors.

For brevity, we omit the finer details of the spot check inner product claims, and point the reader to Section 6.2 for a more detailed overview.

Third step: folding all the claims together. For the purposes of this section, the interaction of $\mathbf{P}$ and $\mathbf{V}$ during the first and second steps can be summarized as follows: by the end of the interaction, $\mathbf{P}$ has sent the oracles $[\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_{\ell_{\boldsymbol{m}}}]\!]$ and $[\![\boldsymbol{w}_1^{\mathsf{aux}}]\!], \ldots, [\![\boldsymbol{w}_{\ell_{\mathsf{aux}}}^{\mathsf{aux}}]\!]$ to $\mathbf{V}$, for some $\ell_{\mathsf{aux}} \in \mathbb{N}$. Additionally, $\mathbf{P}$ and $\mathbf{V}$ have set up numIP inner product claims, where for each $i \in [\mathsf{numIP}]$, $\mathbf{P}$ and $\mathbf{V}$ agree on an oracle $[\![\boldsymbol{w}_i^{\mathsf{IP}}]\!]$ chosen from the set $\big([\![\boldsymbol{w}_1]\!], \ldots, [\![\boldsymbol{w}_{\ell_{\boldsymbol{m}}}]\!], [\![\boldsymbol{w}_1^{\mathsf{aux}}]\!], \ldots, [\![\boldsymbol{w}_{\ell_{\mathsf{aux}}}^{\mathsf{aux}}]\!]\big)$ as well as a vector $\boldsymbol{v}_i^{\mathsf{IP}} \in \mathbb{F}^{k'}$ and a scalar $\sigma_i^{\mathsf{IP}} \in \mathbb{F}$, and set up the claim

$$((\boldsymbol{v}_i^{\mathsf{IP}}, \sigma_i^{\mathsf{IP}}), \boldsymbol{w}_i^{\mathsf{IP}}, \boldsymbol{m}_i^{\mathsf{IP}}) \in_? \mathsf{IP}^{(\mathcal{C}', \varepsilon')} \ ,$$

where $\boldsymbol{m}_i^{\mathsf{IP}} \in \mathbb{F}^{k'}$ is a witness known to $\mathbf{P}$.[9]

The third step of the Reduce IOR involves folding all of these numIP-many $\mathsf{IP}^{(\mathcal{C}', \varepsilon')}$ claims regarding $(\ell_{\boldsymbol{m}} + \ell_{\mathsf{aux}})$-many oracles together into a single $\mathsf{ME}^{(\mathcal{C}', \varepsilon')}$ claim. We note that this step corresponds to the fold step of the IOPP blueprint described in Section 1.2, with $\ell_{\boldsymbol{m}} + \ell_{\mathsf{aux}}$ corresponding to the interleaving factor t.

At a high level, the folding is done as follows: $\mathbf{P}$ and $\mathbf{V}$ run a batched sumcheck protocol to reduce the inner product claims into $\mathsf{ME}^{(\mathcal{C}', \varepsilon')}$ claims (crucially, with respect to the same evaluation point), and then use a random challenge $\beta \xleftarrow{\$} \mathbb{F}$ to $\mathbf{P}$ to batch the $\mathsf{numIP} + \ell_{\boldsymbol{m}}$ many $\mathsf{ME}^{(\mathcal{C}', \varepsilon')}$ claims into a single output $\mathsf{ME}^{(\mathcal{C}', \varepsilon')}$ claim. We refer the reader to Section 6 for the full construction of the Reduce IOR.

[9] We actually also require a few 'triple inner product claims', but omit that detail for now, as handling these is straightforward.

3 Distance analysis of ERA codes

We prove the following theorem about the distance of ERA codes.

Theorem 3.1. *Let $\mathcal{B}$ be a base code with parameters $(\mathbb{F}, n/r, k, \delta_{\mathcal{B}})$, where $r \geq 4$ is a repetition parameter. Let $c > 1$ and $\gamma \in (0, \delta_{\mathcal{B}})$ be constants such that the block length n of $\mathsf{ERA}(\mathbb{F}, \mathcal{B}, r, \boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{M})$ satisfies*

$$\lambda + \log\left(\frac{c^2(r-c)^3}{r^5}\right) \leq \gamma n - 7\log(n+1) \ , \tag{1}$$

where λ is a failure parameter, and the field size q of $\mathbb{F}$ satisfies

$$q - 1 \geq \left(\exp_2\left(\frac{r(2\lambda + 4\log n)}{\delta_{\mathcal{B}} n}\right) \cdot \left(\left(\frac{1}{\delta_{\mathcal{B}}}\right) \cdot e^{(r-r\alpha+1+c\alpha)} \cdot \frac{\left(\frac{r}{c}\right)^{c\alpha}}{(1-\alpha)^{(r-r\alpha)}}\right)\right)^{\frac{1}{c-1}} \ , \tag{2}$$

where $\alpha = 1 - \left(\frac{c}{er}\right)^{c/r}$. Then, the code $\mathsf{ERA}(\mathbb{F}, \mathcal{B}, r, \boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{M})$ has relative distance at least $\mathsf{CalcDist}\,(\lambda, r, \delta_{\mathcal{B}}, n, q)$ with probability at least $1 - 2^{-\lambda}$ over the choice of uniformly sampled permutation matrices $\boldsymbol{P}_1, \boldsymbol{P}_2$ and non-zero diagonal matrix $\boldsymbol{M}$ (see Fig. 1 for the definition of $\mathsf{CalcDist}$).

3.1 Preliminaries

From Section 2, recall that the encoding of a message $\boldsymbol{m} \in \mathbb{F}^k$ under $\mathsf{ERA}(\mathbb{F}, \mathcal{B}, r, \boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{M})$ involves the following steps:

1. The encode step: $\boldsymbol{w}_{\mathcal{B}} := \boldsymbol{G}_{\mathcal{B}} \cdot \boldsymbol{m}$ where $\boldsymbol{G}_{\mathcal{B}} \in \mathbb{F}^{(n/r)\times k}$ is the generator matrix of the base code $\mathcal{B}$.
2. The repeat step: $\boldsymbol{w}_0 := \boldsymbol{R} \cdot \boldsymbol{w}_{\mathcal{B}}$, where $\boldsymbol{R} \in \{0,1\}^{n\times(n/r)}$ is the matrix corresponding to the repetition operation.
3. For $i \in [2]$:
   (a) The permute step: $\boldsymbol{p}_i := \boldsymbol{P}_i \cdot \boldsymbol{w}_{i-1}$, where $\boldsymbol{P}_i \in \{0,1\}^{n\times n}$ is a permutation matrix.
   (b) The multiply step: $\boldsymbol{m}_i := \boldsymbol{M} \cdot \boldsymbol{p}_i$, where $\boldsymbol{M} \in \mathbb{F}^{n\times n}$ is a diagonal matrix with non-zero diagonal entries.
   (c) The accumulate step: $\boldsymbol{w}_i := \boldsymbol{A} \cdot \boldsymbol{m}_i$, where $\boldsymbol{A} \in \{0,1\}^{n\times n}$ is the matrix corresponding to the accumulation / prefix-sum operation.

In the following sections, whenever we consider a fixed message $\boldsymbol{m} \in \mathbb{F}^k$, we will implicitly define $\boldsymbol{w}_{\mathcal{B}}, \boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{w}_2$ relative to $\boldsymbol{m}$ as above.

Definition 3.2 (Configurations)**.** Given a vector $\boldsymbol{w} \in \mathbb{F}^n$, we define its *configuration* $\overline{\boldsymbol{w}} \in \{0,1\}^n$ as the vector that has 1 in the positions where $\boldsymbol{w}$ has non-zero elements, and 0 elsewhere. We use $C_{n,w}$ to denote the set of all configurations $\overline{\boldsymbol{w}} \in \{0,1\}^n$ of weight w. Clearly, $|C_{n,w}| = \binom{n}{w}$, and $\bigcup_{w=0}^{n} C_{n,w} = \{0,1\}^n$.

Definition 3.3 (Intervals)**.** Given $\boldsymbol{w} \in \mathbb{F}^n$ of weight w, let $\alpha_1 < \ldots < \alpha_w$ be the indices of $\boldsymbol{w}$ with non-zero elements.

- The 0-th interval of $\boldsymbol{w}$ is the set of indices $I_0 := [1, 2, \ldots, \alpha_1 - 1]$.
- For each $j \in [w-1]$, the j-th interval of $\boldsymbol{w}$ is the set of indices $I_j := [\alpha_j, \alpha_j + 1, \ldots, \alpha_{j+1} - 1]$.
- The w-th interval of $\boldsymbol{w}$ is the set of indices $I_w := [\alpha_w, \alpha_w + 1, \ldots, n]$.

Definition 3.4 (On-intervals and off-intervals)**.** Given $\boldsymbol{w} \in \mathbb{F}^n$ of weight w, let $\alpha_1 < \ldots < \alpha_w$ be the indices of $\boldsymbol{w}$ that are non-zero and let $I_0, I_1, \ldots, I_w$ be the intervals of $\boldsymbol{w}$. Define $\boldsymbol{a} := \boldsymbol{A}\boldsymbol{w}$. For each $j \in \{0, 1, \ldots, w\}$, the entries of $\boldsymbol{a}$ in the indices of I_j are all equal to the same value. In particular, this value is 0 for I_0 and $\sum_{k=1}^{j} \boldsymbol{w}[\alpha_k]$, for all $j \in [w]$. For each $j \in \{0, 1, \ldots, w\}$, we say that the j-th interval I_j of $\boldsymbol{w}$ is an *on-interval* if $\boldsymbol{a}$ has non-zero values in the indices of I_j; otherwise, we say that I_j is an *off-interval*. Note that the 0-th interval is always an off-interval.

Definition 3.5 (Viable set of on-intervals)**.** Given a configuration $\overline{\boldsymbol{w}} \in C_{n,w}$, we define its intervals $I_0, I_1, \ldots, I_w$ in the same way as in Definition 3.3. We say a subset of the intervals $\{I_j\}_{j=0}^{w}$ is *viable for* $\overline{\boldsymbol{w}}$ if there exists a vector $\boldsymbol{w} \in \mathbb{F}^n$ of configuration $\overline{\boldsymbol{w}}$ such that the intervals in the subset are exactly the on-intervals of $\boldsymbol{w}$.

Fact 3.6 (Entropy bound)**.** For any $k, n \in \mathbb{N}$ such that $k \leq n$, we have

$$\frac{\exp_2(n \cdot H(k/n))}{n+1} \leq \binom{n}{k} \leq \exp_2(n \cdot H(k/n)) \ ,$$

where $H(p) = -p \log_2 p - (1-p)\log_2(1-p)$ is the binary entropy function.

Event definitions. Recall that $r \in \mathbb{N}$ is a repetition parameter, $c > 1$ is a fixed constant, and λ is a soundness parameter. Given an input message $\boldsymbol{m} \in \mathbb{F}^k$, recall that $\boldsymbol{w}_0 := \boldsymbol{R}\boldsymbol{G}_{\mathcal{B}}\boldsymbol{m}$, $\boldsymbol{w}_1 := \boldsymbol{A}\boldsymbol{M}\boldsymbol{P}_1\boldsymbol{w}_0$, and $\boldsymbol{w}_2 := \boldsymbol{A}\boldsymbol{M}\boldsymbol{P}_2\boldsymbol{w}_1$. Define the following events:

1. $\mathcal{F}_{\mathsf{mult1}}$: there exists $\boldsymbol{m} \in \mathbb{F}^k$ such that the number of on-intervals in $\boldsymbol{M}\boldsymbol{P}_1\boldsymbol{w}_0$ is $\leq \mathsf{wt}(\boldsymbol{w}_0)(1 - c/r)$. This event captures the case where the first multiplication vector is 'bad', and thus the probability of this event is analyzed over the randomness of $\boldsymbol{M}$.

2. $\mathcal{F}_{\mathsf{mult2}}$: there exists $\boldsymbol{m} \in \mathbb{F}^k$ such that the number of on-intervals in $\boldsymbol{M}\boldsymbol{P}_2\boldsymbol{w}_1$ is $\leq \mathsf{wt}(\boldsymbol{w}_1)(1 - c/r)$. The probability of this event is also analyzed over the randomness of $\boldsymbol{M}$.

3. $\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta)$: there exists $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = \ell_0$, $\mathsf{wt}(\boldsymbol{w}_1) = \ell_1$, and $\mathsf{wt}(\boldsymbol{w}_2) < \delta n$. Additionally, define $\mathcal{F}_{\mathsf{perm}}(\delta)$ as the event that occurs if there exists $\ell_0 \in [\delta_{\mathcal{B}} n, n]$ and $\ell_1 \in [\lceil \ell_0(1 - c/r) \rceil, n]$ such that $\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta)$ occurs. This event captures the case where the pair of permutations is 'bad', and thus the probability of this event is analyzed over the randomness of $\boldsymbol{P}_1, \boldsymbol{P}_2$.

If $\mathcal{F}_{\mathsf{perm}}(\delta)$ does not occur then $\mathsf{ERA}(\mathbb{F}, \mathcal{B}, r, \boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{M})$ has relative distance at least δ. Applying law of total probability, we have

$$\Pr(\mathcal{F}_{\mathsf{perm}}(\delta)) \leq \Pr(\mathcal{F}_{\mathsf{mult1}}) + \Pr(\mathcal{F}_{\mathsf{mult2}}) + \Pr(\mathcal{F}_{\mathsf{perm}}(\delta) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) \ .$$

Thus, we will focus on upper bounding these three terms in the following sections by $2^{-\lambda-2}, 2^{-\lambda-2}$, and $2^{-\lambda-1}$ respectively.

3.2 Upper bounding $\Pr(\mathcal{F}_{\mathsf{mult1}})$

We begin by proving the following lemma.

Lemma 3.7 (Large fraction of intervals are on)**.** *Consider a fixed vector $\boldsymbol{x} \in \mathbb{F}^n$ of weight w. Let $\eta \in (0, 1)$ be a constant, and let $\boldsymbol{M} \in \mathbb{F}^{n \times n}$ be a diagonal matrix where each diagonal entry is sampled independently and uniformly at random from $\mathbb{F} \setminus \{0\}$. Then we have that*

$$\Pr\left[\textit{Number of on-intervals in } \boldsymbol{M}\boldsymbol{x} \textit{ is } \leq w \cdot \eta\right] \leq \left(\frac{e}{(1-\eta)(q-1)}\right)^{w(1-\eta)} \ .$$

Proof. Let the non-zero indices of $\boldsymbol{x}$ be $\alpha_1, \ldots, \alpha_w$. As noted in Definition 3.4, for each $j \in [w]$, the value $\boldsymbol{AMx}[i]$ must be identical for all indices $i \in I_j$, and we denote this value by e_j. For each $j \in [w]$, define the Bernoulli random variable X_j to be 1 if and only if $e_j = 0$, and define $X := \sum_{j \in [w]} X_j$. Thus, X is the random variable for the number of off-intervals in $\boldsymbol{Mx}$ (not including the 0-th interval). We have

$$\Pr[X_j = 1] = \begin{cases} 0 & \text{if } j = 1 \\ 0 & \text{if } j > 1 \text{ and } X_{j-1} = 1 \\ \frac{1}{q-1} & \text{if } j > 1 \text{ and } X_{j-1} = 0 \end{cases} .$$

That is, the first interval is always an on-interval, and an off-interval is always followed by an on-interval. Additionally, if the $(j-1)$-th interval is an on-interval, then the j-th interval is an off-interval with probability $1/(q-1)$. This can be seen as follows: let $\boldsymbol{v}$ be the vector contained in the diagonal indices of $\boldsymbol{M}$. We have $e_j = 0$ only when $\boldsymbol{Mx}[\alpha_j] = -(\sum_{i=1}^{\alpha_j - 1} \boldsymbol{Mx}[i])$, which can be rewritten as $\boldsymbol{v}[\alpha_j] \cdot \boldsymbol{x}[\alpha_j] = -(\sum_{i=1}^{\alpha_j - 1} \boldsymbol{v}[i] \cdot \boldsymbol{x}[i])$. Fix all the entries of $\boldsymbol{v}$ except for $\boldsymbol{v}[\alpha_j]$, then there is exactly one value of $\boldsymbol{v}[\alpha_j]$ that satisfies this equation.

Now, for each $j \in [w]$, define Y_j as i.i.d. Bernoulli random variables that are 1 with probability $1/(q-1)$. If $Y := \sum_{j \in [w]} Y_j$, then $E[Y] = w/(q-1)$, and $\Pr[X > t] \leq \Pr[Y > t]$ for any $t \geq 0$. We can use the following multiplicative Chernoff bound where $\delta > 0$:

$$\Pr[X > (1+\delta)\mathbb{E}[Y]] \leq \Pr[Y > (1+\delta)\mathbb{E}[Y]] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}} \right)^{\mathbb{E}[Y]} .$$

In particular, set $\delta = (1-\eta)(q-1) - 1$. Thus, $(1+\delta)\mathbb{E}[Y] = w(1-\eta)$ and we have

$$\begin{aligned} \Pr[X > w(1-\eta)] &\leq \left(\frac{e^{(1-\eta)(q-1)-1}}{((1-\eta)(q-1))^{(1-\eta)(q-1)}} \right)^{w/(q-1)} \\ &\leq \left(\frac{e}{(1-\eta)(q-1)} \right)^{(1-\eta)(q-1)\cdot(w/(q-1))} = \left(\frac{e}{(1-\eta)(q-1)} \right)^{w(1-\eta)} . \square \end{aligned}$$

Lemma 3.8 (Probability of $\mathcal{F}_{\mathsf{mult1}}$)**.** *We have* $\Pr(\mathcal{F}_{\mathsf{mult1}}) \leq 2^{-\lambda-2}$ *if*

$$q - 1 \geq \left(\frac{e}{\delta_{\mathcal{B}}} \cdot \left(\frac{er}{c} \right)^c \cdot \exp_2 \left(\frac{r(\lambda + 2 + \log n)}{\delta_{\mathcal{B}} n} \right) \right)^{\frac{1}{c-1}} .$$

Proof. Consider a fixed $w \in [\lceil \delta_{\mathcal{B}} n \rceil, n]$. Define $\mathcal{F}_{\mathsf{mult1}}(w)$ as the event that there exists $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = w$ and the number of on-intervals in $\boldsymbol{MP}_1\boldsymbol{w}_0$ is $\leq w(1 - c/r)$. If we can show that $\Pr(\mathcal{F}_{\mathsf{mult1}}(w)) \leq 2^{-\lambda-2}/n$, then applying the union bound over all such w gives the desired result. Thus, we focus on upper bounding $\Pr(\mathcal{F}_{\mathsf{mult1}}(w))$.

The number of messages $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = w$ is

$$\leq \binom{n/r}{w/r} \cdot (q-1)^{w/r} .$$

This is because there are at most $\binom{n/r}{w/r}$ ways to choose which indices are non-zero in $\boldsymbol{w}_0$, and for each such choice, there are at most $(q-1)^{w/r}$ ways to assign non-zero values to these indices.

Now, substituting η with $(1 - c/r)$ in Lemma 3.7, we get

$$\Pr\left[\text{Number of on-intervals in } \boldsymbol{M}\boldsymbol{P}_1\boldsymbol{w}_0 \text{ is } \leq w \cdot (1 - c/r)\right] \leq \left(\frac{er}{c(q-1)}\right)^{cw/r} .$$

Applying the union bound over all messages $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = w$, we get

$$\begin{aligned}
\Pr(\mathcal{F}_{\mathsf{mult1}}(w)) &\leq \binom{n/r}{w/r} \cdot (q-1)^{w/r} \cdot \left(\frac{er}{c(q-1)}\right)^{cw/r} \\
&\leq \left(\frac{en}{w}\right)^{w/r} \cdot (q-1)^{w/r} \cdot \left(\frac{er}{c(q-1)}\right)^{cw/r} \\
&= \left(\frac{en}{w} \cdot \left(\frac{er}{c}\right)^{c} \cdot (q-1)^{1-c}\right)^{w/r} .
\end{aligned}$$

Thus, if we want $\Pr(\mathcal{F}_{\mathsf{mult1}}(w)) \leq 2^{-\lambda-2}/n$, it suffices to have

$$\begin{aligned}
&\frac{en}{w} \cdot \left(\frac{er}{c}\right)^{c} \cdot (q-1)^{1-c} \leq \exp_2\left(-(\lambda + \log n)r/w\right) \\
\iff &\left(\frac{en}{w} \cdot \left(\frac{er}{c}\right)^{c} \cdot \exp_2\left(\frac{r(\lambda + 2 + \log n)}{w}\right)\right)^{\frac{1}{c-1}} \leq q - 1 .
\end{aligned}$$

Since $w \geq \delta_{\mathcal{B}} n$, we get the desired bound on q. □

Remark 3.9 (Lower bound on c)**.** Clearly, $c > 1$ is necessary for the bounds in Lemma 3.8 to hold. Thus, we cannot show that more than $(1 - 1/r)$-fraction of intervals are on with high probability using this proof technique, regardless of the choice of q.

3.3 Upper bounding $\Pr(\mathcal{F}_{\mathsf{mult2}})$

It might appear that proving $\Pr(\mathcal{F}_{\mathsf{mult2}}) \leq 2^{-\lambda-2}$ is nearly identical to proving Lemma 3.8. However, there is a subtle challenge that we need to overcome. We upper bounded $\Pr(\mathcal{F}_{\mathsf{mult1}})$ by introducing the event $\mathcal{F}_{\mathsf{mult1}}(w)$ for a fixed weight w of $\boldsymbol{w}_0$. We can similarly introduce the event $\mathcal{F}_{\mathsf{mult2}}(w_0, w_1)$, which occurs if there exists $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = w_0$ and $\mathsf{wt}(\boldsymbol{w}_1) = w_1$ and the number of on-intervals in $\boldsymbol{M}\boldsymbol{P}_2\boldsymbol{w}_1$ is $\leq w_1(1 - c/r)$. By Lemma 3.7, for a fixed message the probability of this occurring is $\leq (er/(c(q-1)))^{cw_1/r}$. Since we need to apply a union bound over $\binom{n/r}{w_0/r}(q-1)^{w_0/r}$ input messages, the subtlety is that we need to examine how w_1 compares to w_0. Thus, we consider two cases:

1. If $w_1 \geq w_0$, we upper bound $\Pr(\mathcal{F}_{\mathsf{mult2}}(w_0, w_1))$ by using the same approach as in Lemma 3.8.
2. If $w_1 < w_0$, we proceed as follows:
   (a) For any $w_0 \in [\lceil \delta_{\mathcal{B}} n \rceil, n]$, prove the following holds with overwhelming probability: there are at most $\varphi(w_0, t)$ messages $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = w_0$ and $\mathsf{wt}(\boldsymbol{w}_1) = w_0 \cdot t$ where $t \in (0, 1)$ and φ is some function. We refer to these messages as *shrinking* messages.
   (b) Apply the union bound over these $\varphi(w_0, t)$ shrinking messages to prove the following holds with overwhelming probability: for all messages $\boldsymbol{m}$ such that $\mathsf{wt}(\boldsymbol{w}_0) = w_0$ and $\mathsf{wt}(\boldsymbol{w}_1) = w_0 \cdot t$ where $t \in (0, 1)$, the number of on-intervals in $\boldsymbol{M}\boldsymbol{P}_2\boldsymbol{w}_1$ is $> w_1(1 - c/r)$.

Proving these two statements requires a lower bound for q in terms of w_0 and t. Our final lower bound for q is the largest lower bound obtained over the choice of w_0 and t.

Lemma 3.10 (Upper bound on number of shrinking messages)**.** *Let $w_0 \in [\lceil \delta_{\mathcal{B}} n \rceil, n]$ and $t \in (0,1)$. Define*

$$\varphi(w_0, t) := 2^{-\lambda-2} \cdot n^{-2} \cdot \left(\frac{c(q-1)}{er}\right)^{cw_0 t/r}$$

Then, the probability that there are more than $\varphi(w_0, t)$ messages $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = w_0$ and $\mathsf{wt}(\boldsymbol{w}_1) = w_0 \cdot t$ is at most $2^{-\lambda-2} \cdot n^{-2}$ if

$$q - 1 \geq \left(\left(2^{2\lambda+4} \cdot n^4\right)^{r/w_0} \cdot \left(\left(\frac{n}{w_0}\right) \cdot e^{(r-r\alpha+1+c\alpha)} \cdot \frac{\left(\frac{r}{c}\right)^{c\alpha}}{(1-\alpha)^{(r-r\alpha)}}\right)\right)^{\frac{1}{c-1}} , \tag{3}$$

where $\alpha = 1 - \left(\frac{c}{er}\right)^{c/r}$.

Proof. Let $X_{\boldsymbol{m}}$ be a Bernoulli random variable that is 1 if and only if $\boldsymbol{m} \in \mathbb{F}^k$ satisfies $\mathsf{wt}(\boldsymbol{w}_0) = w_0$ and $\mathsf{wt}(\boldsymbol{w}_1) \leq w_0 \cdot t$. By Lemma 3.7, we have

$$\Pr(X_{\boldsymbol{m}} = 1) \leq \left(\frac{e}{(1-t)(q-1)}\right)^{w_0(1-t)} .$$

Let $X := \sum_{\boldsymbol{m} \in \mathbb{F}^k : \mathsf{wt}(\boldsymbol{w}_0) = w_0} X_{\boldsymbol{m}}$. Then we have

$$E[X] \leq \binom{n/r}{w_0/r} \cdot (q-1)^{w_0/r} \cdot \left(\frac{e}{(1-t)(q-1)}\right)^{w_0(1-t)} \leq \left(\frac{en(q-1)}{w_0}\right)^{w_0/r} \cdot \left(\frac{e}{(1-t)(q-1)}\right)^{w_0(1-t)}$$

By Markov's inequality, we have

$$\Pr\left(X > \varphi(w_0, t)\right) \leq \frac{E[X]}{\varphi(w_0, t)} \leq \frac{\left(\frac{en(q-1)}{w_0}\right)^{w_0/r} \cdot \left(\frac{e}{(1-t)(q-1)}\right)^{w_0(1-t)}}{2^{-\lambda-2} \cdot n^{-2} \cdot \left(\frac{c(q-1)}{er}\right)^{cw_0 t/r}}$$

We want this probability to be at most $2^{-\lambda-2} \cdot n^{-2}$, which holds if

$$(q-1)^{w_0(1-t-1/r+ct/r)} \geq 2^{2\lambda+4} \cdot n^4 \cdot \left(\frac{n}{w_0}\right)^{w_0/r} \cdot \left(e^{(1-t+1/r+ct/r)} \cdot \frac{\left(\frac{r}{c}\right)^{ct/r}}{(1-t)^{(1-t)}}\right)^{w_0} . \tag{4}$$

Define

$$g(t) := \ln\left(e^{(1-t+1/r+ct/r)} \cdot \frac{\left(\frac{r}{c}\right)^{ct/r}}{(1-t)^{(1-t)}}\right) = (1 - t + 1/r + ct/r) + (ct/r) \cdot \ln\left(\frac{r}{c}\right) - (1-t)\ln(1-t) .$$

Then $g'(t) = -1 + \frac{c}{r} + \frac{c}{r}\ln\left(\frac{r}{c}\right) + 1 + \ln(1-t) = \frac{c}{r}\ln\left(\frac{er}{c}\right) + \ln(1-t)$. Clearly, $g'(t)$ is decreasing in t, so $g(t)$ is maximized at a point α such that $g'(\alpha) = 0$, i.e., $\alpha = 1 - \left(\frac{c}{er}\right)^{c/r}$. Since $\exp$ is a strictly increasing function, $\exp(g(t))$ is also maximized at $t = \alpha$. Substituting $t = \alpha$ in (4), we get

$$(q-1)^{w_0(1-t-1/r+ct/r)} \geq 2^{2\lambda+4} \cdot n^4 \cdot \left(\frac{n}{w_0}\right)^{w_0/r} \cdot \left(e^{(1-\alpha+1/r+c\alpha/r)} \cdot \frac{\left(\frac{r}{c}\right)^{c\alpha/r}}{(1-\alpha)^{(1-\alpha)}}\right)^{w_0} .$$

The exponent on the left-hand side is minimized at $t = 1$, which finally gives us

$$(q-1)^{w_0((c-1)/r)} \geq 2^{2\lambda+4} \cdot n^4 \cdot \left(\frac{n}{w_0}\right)^{w_0/r} \cdot \left(e^{(1-\alpha+1/r+c\alpha/r)} \cdot \frac{\left(\frac{r}{c}\right)^{c\alpha/r}}{(1-\alpha)^{(1-\alpha)}}\right)^{w_0} .$$

Rearranging the terms, we get that if the lower bound on q in (3) holds, then (4) also holds for any $t \in (0,1)$, which completes the proof. □

Lemma 3.11 (Probability of $\mathcal{F}_{\mathsf{mult2}}$)**.** *We have* $\Pr(\mathcal{F}_{\mathsf{mult2}}) \leq 2^{-\lambda-2}$ *if*

$$q-1 \geq \left(\exp_2\left(\frac{r(2\lambda+4+4\log n)}{\delta_{\mathcal{B}} n}\right) \cdot \left(\left(\frac{1}{\delta_{\mathcal{B}}}\right) \cdot e^{(r-r\alpha+1+c\alpha)} \cdot \frac{\left(\frac{r}{c}\right)^{c\alpha}}{(1-\alpha)^{(r-r\alpha)}}\right)\right)^{\frac{1}{c-1}} ,$$

where $\alpha = 1 - \left(\frac{c}{er}\right)^{c/r}$.

Proof. Consider a fixed $w_0 \in [\lceil \delta_{\mathcal{B}} n \rceil, n]$ and $w_1 \in [1, n]$. Define $\mathcal{F}_{\mathsf{mult2}}(w_0, w_1)$ as the event that there exists $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = w_0$ and $\mathsf{wt}(\boldsymbol{w}_1) = w_1$ and the number of on-intervals in $\boldsymbol{M}\boldsymbol{P}_2\boldsymbol{w}_1$ is $\leq w_1(1-c/r)$. If we can show that $\Pr(\mathcal{F}_{\mathsf{mult2}}(w_0, w_1)) \leq 2^{-\lambda} \cdot n^{-2}$, then applying the union bound over all such w_0, w_1 gives the desired result. Thus, we focus on upper bounding $\Pr(\mathcal{F}_{\mathsf{mult2}}(w_0, w_1))$. We consider two cases based on the relationship between w_0 and w_1.

1. $w_1 \geq w_0$. In this case, we proceed as in Lemma 3.8. By plugging in $\eta = (1 - c/r)$ in Lemma 3.7, we get that

   $$\Pr\left[\text{Number of on-intervals in } \boldsymbol{M}\boldsymbol{P}_2\boldsymbol{w}_1 \text{ is } \leq w_1(1-c/r)\right] \leq \left(\frac{er}{c(q-1)}\right)^{cw_1/r} .$$

   Applying the union bound over all messages $\boldsymbol{m} \in \mathbb{F}^k$ such that $\boldsymbol{w}_0 := \boldsymbol{R}\boldsymbol{G}_{\mathcal{B}}\boldsymbol{m}$ has $\mathsf{wt}(\boldsymbol{w}_0) = w_0$, we get

   $$\begin{aligned}\Pr(\mathcal{F}_{\mathsf{mult2}}(w_0, w_1)) &\leq \binom{n/r}{w_0/r} \cdot (q-1)^{w_0/r} \cdot \left(\frac{er}{c(q-1)}\right)^{cw_1/r} \\ &\leq \left(\frac{en}{w_0}\right)^{w_0/r} \cdot (q-1)^{w_0/r} \cdot \left(\frac{er}{c(q-1)}\right)^{cw_0/r} .\end{aligned}$$

   Similar to Lemma 3.8, if we want $\Pr(\mathcal{F}_{\mathsf{mult2}}(w_0, w_1)) \leq 2^{-\lambda-2} \cdot n^{-2}$, it suffices to have

   $$\begin{aligned}&\frac{en}{w_0} \cdot \left(\frac{er}{c}\right)^c \cdot (q-1)^{1-c} \leq \exp_2\left(-(\lambda+2+2\log n)r/w_0\right) \\ \iff &\left(\frac{en}{w_0} \cdot \left(\frac{er}{c}\right)^c \cdot \exp_2\left(\frac{r(\lambda+2+2\log n)}{w_0}\right)\right)^{\frac{1}{c-1}} \leq q-1 .\end{aligned}$$

   Replacing w_0 with the smallest possible value $\lceil \delta_{\mathcal{B}} n \rceil$ gives a sufficient condition for the above inequality to hold for any w_0 in the specified range.

2. $w_1 < w_0$. Let $t := w_1/w_0 \in (0,1)$. By Lemma 3.10, the probability that there are more than $\varphi(w_0, t)$ messages $\boldsymbol{m} \in \mathbb{F}^k$ such that $\mathsf{wt}(\boldsymbol{w}_0) = w_0$ and $\mathsf{wt}(\boldsymbol{w}_1) = w_0 \cdot t$ is at most $2^{-\lambda-2} \cdot n^{-2}$ if the lower bound on q in (3) holds. Applying the union bound over all such $\varphi(w_0, t)$ shrinking messages and using Lemma 3.7 with $\eta = (1 - c/r)$, we get

   $$\Pr(\mathcal{F}_{\mathsf{mult2}}(w_0, w_1)) \leq \varphi(w_0, t) \cdot \left(\frac{er}{c(q-1)}\right)^{cw_1/r}$$

$$= 2^{-\lambda-2} \cdot n^{-2} \cdot \left(\frac{c(q-1)}{er}\right)^{cw_0 t/r} \cdot \left(\frac{er}{c(q-1)}\right)^{cw_1/r}$$
$$= 2^{-\lambda-2} \cdot n^{-2} \ .$$

Since the lower bound on q in (3) (when w_0 is replaced with the worst case value of $\delta_{\mathcal{B}} n$) is larger than the lower bound obtained in the first case, we get the desired result. □

3.4 Upper bounding $\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}})$

Recall that $\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta)$ is the event that there exists some message $\boldsymbol{m} \in \mathbb{F}^k$ such that the following three conditions hold: (a) $\mathsf{wt}(\boldsymbol{w}_0) = \ell_0$, (b) $\mathsf{wt}(\boldsymbol{w}_1) = \ell_1$, (c) $\mathsf{wt}(\boldsymbol{w}_2) < \delta n$. We analyze three cases based on the values of ℓ_0 and ℓ_1.

Large input weight. The first case is when the input weight $\ell_0 \geq \lfloor \delta n/(1-c/r)^2 \rfloor + 1$.

Lemma 3.12. *Given $\delta < (1-c/r)^2$ and $\ell_0 \in \left[\lfloor \delta n/(1-c/r)^2 \rfloor + 1, n\right]$, for any $\ell_1 \in [0, n]$, we have*

$$\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) = 0 \ .$$

Proof. Consider a message $\boldsymbol{m} \in \mathbb{F}^k$ such that $\boldsymbol{w}_0$ has weight ℓ_0. Since $\mathcal{F}_{\mathsf{mult1}}$ does not occur, more than $\ell_0(1-c/r)$ intervals of $\boldsymbol{M}\boldsymbol{P}_1\boldsymbol{w}_0$ are on-intervals, and therefore $\mathsf{wt}(\boldsymbol{w}_1) > \ell_0(1-c/r)$. Similarly, since $\mathcal{F}_{\mathsf{mult2}}$ does not occur, more than $\mathsf{wt}(\boldsymbol{w}_1)(1-c/r)$ intervals of $\boldsymbol{M}\boldsymbol{P}_2\boldsymbol{w}_1$ are on-intervals, and therefore

$$\mathsf{wt}(\boldsymbol{w}_2) > \left(\lfloor \delta n/(1-c/r)^2 \rfloor + 1\right)(1-c/r)^2 \geq \delta n \ .$$

This completes the proof. □

Large intermediate weight. The second case is when the intermediate weight $\ell_1 \geq \lfloor \delta n/(1-c/r) \rfloor + 1$. The proof of the following lemma is analogous to that of Lemma 3.12.

Lemma 3.13. *Given $\delta < (1-c/r)$ and $\ell_1 \in [\lfloor \delta n/(1-c/r) \rfloor + 1, n]$, for any $\ell_0 \in [0, n]$, we have*

$$\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) = 0 \ .$$

Before discussing the final case (where both ℓ_0 and ℓ_1 are small), we first prove the following lemma about viable sets of on-intervals (Definition 3.5).

Lemma 3.14. *Define $A_{n,\ell_0,\ell_1,m} \subseteq C_{n,\ell_0}$ as the set of configurations such that $\overline{\boldsymbol{w}} \in A_{n,\ell_0,\ell_1,m}$ if and only if there exists a viable set of on-intervals S for $\overline{\boldsymbol{w}}$ such that $|S| = m$, and the total length of the on-intervals is ℓ_1. Then, we have*

$$|A_{n,\ell_0,\ell_1,m}| \leq \binom{m}{\ell_0 - m} \cdot \binom{\ell_1}{m} \cdot \binom{n-\ell_1}{\ell_0 - m} \ .$$

Proof. The proof extends ideas used in prior work [DJM98; BFKTWZ24; BCFRRZ25]. Firstly, if S is a viable set of on-intervals for a configuration $\overline{\boldsymbol{w}}$ with intervals $\{I_j\}_{j=0}^{\ell_0}$, then $\{I_j\}_{j=0}^{\ell_0} \setminus S$ does not contain any consecutive intervals, i.e., no two off-intervals are adjacent.

Any configuration $\overline{\boldsymbol{w}} \in A_{n,\ell_0,\ell_1,m}$ is uniquely defined by its non-zero indices $\alpha_1 < \alpha_2 < \ldots < \alpha_{\ell_0}$. The number of choices for such indices (that also satisfy the conditions for $A_{n,\ell_0,\ell_1,m}$) can be upper bounded as follows:

1. Choose the m on-intervals out of the ℓ_0 intervals such that no two off-intervals are adjacent. This is a stars-and-bars problem, where the stars are the m on-intervals and the bars are the $\ell_0 - m$ off-intervals. In particular, the m stars induce m gaps between them (including the gap after the last star) where we can place the remaining $\ell_0 - m$ bars (since no two off-intervals can be adjacent, there must be at least one on-interval between any two off-intervals). Thus, this can be done in $\binom{m}{\ell_0 - m}$ ways.

2. Choose the lengths of the on-intervals such that the sum of their lengths is exactly ℓ_1. This is a stars-and-bars problem with ℓ_1 stars and $m - 1$ bars. Since each on-interval must have length at least 1, there are $\ell_1 - 1$ choices for placing the $m - 1$ bars. Thus, this can be done in $\binom{\ell_1 - 1}{m-1} \leq \binom{\ell_1}{m}$ ways.

3. Choose the lengths of the remaining $\ell_0 - m + 1$ off-intervals such that the sum of their lengths is $n - \ell_1$ (this includes the 0-th interval). This is also a stars-and-bars problem, where there are $n - \ell_1$ stars and $\ell_0 - m$ bars. The stars induce $n - \ell_1$ gaps between them (including the gap before the first star, since the 0-th interval can have length 0, but not a gap after the last star since other intervals have length at least 1), where we can place the $\ell_0 - m$ bars. Thus, this can be done in $\binom{n-\ell_1}{\ell_0 - m}$ ways.

This completes the proof. □

Corollary 3.15. *Let $A_{n,\ell_0,\ell_1,m} \subseteq C_{n,\ell_0}$ be as defined in Lemma 3.14. Then, $|A_{n,\ell_0,\ell_1,m}| \leq \exp_2(g(n, \ell_0, \ell_1, m))$ where*

$$g(n, \ell_0, \ell_1, m) := m \cdot H\left(\frac{\ell_0 - m}{m}\right) + \ell_1 \cdot H\left(\frac{m}{\ell_1}\right) + (n - \ell_1) \cdot H\left(\frac{\ell_0 - m}{n - \ell_1}\right) \quad . \tag{5}$$

Proof. Follows directly from Lemma 3.14 and Fact 3.6. □

Small input and intermediate weight. The final (and most involved) case is when $\ell_0 \leq \lfloor \delta n/(1 - c/r)^2 \rfloor$ and $\ell_1 \leq \lfloor \delta n/(1 - c/r) \rfloor$ simultaneously. We analyze this case in the following lemma.

Lemma 3.16. *Let $r \geq 4$ be a repetition parameter, let $c \in (1, r/2 \cdot (1 - 1/\sqrt{5}))$, and let $\delta < (1 - c/r)^2$. Given $\ell_0 \leq \lfloor \delta n/(1 - c/r)^2 \rfloor$, and $\ell_1 \leq \lfloor \delta n/(1 - c/r) \rfloor$, define $d_0 := \ell_0/n$, $d_1 := \ell_1/n$. Additionally, define the following two values:*

$$\textbf{\textit{Polynomial term:}}\ P_{\mathsf{perm}}(\delta) := \frac{c^2 r \delta^3 n^3 (n+1)^2}{(r-c)^3} \quad , \tag{6}$$

$$\begin{aligned}\textbf{\textit{Exponent:}}\ E_{\mathsf{perm}}(d_0, d_1, \delta) = n \cdot \Bigg(& \frac{1}{r} H(d_0) \\ & + d_0(1 - c/r) H\left(\frac{c}{r-c}\right) + d_1 H\left(\frac{d_0(1-c/r)}{d_1}\right) + (1 - d_1) H\left(\frac{d_0 c}{r(1-d_1)}\right) - H(d_0) \\ & + d_1(1 - c/r) H\left(\frac{c}{r-c}\right) + \delta H\left(\frac{d_1(1-c/r)}{\delta}\right) + (1 - \delta) H\left(\frac{d_1 c}{r(1-\delta)}\right) - H(d_1) \Bigg) \quad . \end{aligned} \tag{7}$$

Then, we have $\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) \leq P_{\mathsf{perm}}(\delta) \cdot \exp_2(E_{\mathsf{perm}}(d_0, d_1, \delta))$.

Proof. Fix a particular configuration $\overline{\boldsymbol{w}}_0 \in C_{n,\ell_0}$ that can be obtained at the end of the repeat step. Clearly, the number of such configurations is $\binom{n/r}{\ell_0/r}$. Recall that $\boldsymbol{w}_0 := \boldsymbol{R} \cdot \boldsymbol{G}_{\mathcal{B}} \cdot \boldsymbol{m}$, $\boldsymbol{m}_1 := \boldsymbol{M} \cdot \boldsymbol{P}_1 \cdot \boldsymbol{w}_0$, $\boldsymbol{w}_1 := \boldsymbol{A} \cdot \boldsymbol{m}_1$, $\boldsymbol{m}_2 := \boldsymbol{M} \cdot \boldsymbol{P}_2 \cdot \boldsymbol{w}_1$, and $\boldsymbol{w}_2 := \boldsymbol{A} \cdot \boldsymbol{m}_2$. Define $\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \overline{\boldsymbol{w}}_0, \delta)$ as the event that there exists a message

$\boldsymbol{m} \in \mathbb{F}^k$ such that: (a) $\overline{\boldsymbol{w}}_0$ is the configuration of $\boldsymbol{w}_0$, (b) $\mathsf{wt}(\boldsymbol{w}_1) = \ell_1$, and (c) $\mathsf{wt}(\boldsymbol{w}_2) < \delta n$. Looking ahead, we will apply a union bound over all $\binom{n/r}{\ell_0/r}$ such configurations $\overline{\boldsymbol{w}}_0$ to obtain an upper bound on $\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta) \mid \neg\mathcal{F}_{\mathsf{mult2}}, \neg\mathcal{F}_{\mathsf{mult1}})$.

Upper bounding failure probability for fixed $\overline{\boldsymbol{w}}_0$. Since we condition on $\mathcal{F}_{\mathsf{mult1}}$ not occurring, the number of on-intervals in $\boldsymbol{m}_1$ is $> \ell_0(1 - c/r)$. Similarly, since we condition on $\mathcal{F}_{\mathsf{mult2}}$ not occurring, the number of on-intervals in $\boldsymbol{m}_2$ is $> \ell_1(1 - c/r)$. Consider a malicious adversary $\mathcal{A}$ who wants to cause $\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \overline{\boldsymbol{w}}_0, \delta)$ to occur despite $\mathcal{F}_{\mathsf{mult1}}$ and $\mathcal{F}_{\mathsf{mult2}}$ not occurring. $\mathcal{A}$ follows the strategy below.

1. Since $\overline{\boldsymbol{w}}_0$ is fixed, so is the configuration $\overline{\boldsymbol{m}}_1$ of $\boldsymbol{m}_1$. Note that $\boldsymbol{m}_1$ itself is not fixed, so $\mathcal{A}$ has the freedom to choose how many and which intervals of $\boldsymbol{m}_1$ are on-intervals:
   (a) $\mathcal{A}$ chooses $m_1 \in [\lfloor \ell_0(1 - c/r) \rfloor + 1, \ell_0]$, which corresponds to the number of on-intervals in $\overline{\boldsymbol{m}}_1$.
   (b) $\mathcal{A}$ chooses a viable set of m_1 on-intervals for $\overline{\boldsymbol{m}}_1$ (see Definition 3.5) such that the sum of lengths of the chosen on-intervals is ℓ_1.
2. Since the on-intervals for $\boldsymbol{m}_1$ were fixed in the previous step, the configuration $\overline{\boldsymbol{w}}_1$ of $\boldsymbol{w}_1$ is also fixed, which in turn fixes the configuration $\overline{\boldsymbol{m}}_2$ of $\boldsymbol{m}_2$. Again, $\boldsymbol{m}_2$ itself is not fixed, so $\mathcal{A}$ has the freedom to choose how many and which intervals of $\boldsymbol{m}_2$ are on-intervals:
   (a) $\mathcal{A}$ chooses $m_2 \in [\lfloor \ell_1(1 - c/r) \rfloor + 1, \ell_1]$, which corresponds to the number of on-intervals in $\overline{\boldsymbol{m}}_2$.
   (b) $\mathcal{A}$ chooses a viable set of m_2 on-intervals for $\overline{\boldsymbol{m}}_2$ such that the sum of lengths of the chosen on-intervals is some $\ell_2 < \delta n$.

Clearly, $\mathcal{A}$ succeeds if the following two conditions hold:

1. There exists a viable set of on-intervals for $\overline{\boldsymbol{m}}_1$ such that the number of on-intervals is m_1, and the sum of lengths of the chosen on-intervals is ℓ_1. The number of such configurations $\overline{\boldsymbol{m}}_1$ is $|A_{n,\ell_0,\ell_1,m_1}|$ (see Lemma 3.14).
2. There exists a viable set of on-intervals for $\overline{\boldsymbol{m}}_2$ such that the number of on-intervals is m_2, and the sum of lengths of the chosen on-intervals is $\ell_2 < \delta n$. The number of such configurations $\overline{\boldsymbol{m}}_2$ is $|A_{n,\ell_1,\ell_2,m_2}|$ (see Lemma 3.14).

Therefore, to upper bound the probability of $\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \overline{\boldsymbol{w}}_0, \delta)$ occurring, it suffices to upper bound the number of pairs of configurations $(\overline{\boldsymbol{m}}_1, \overline{\boldsymbol{m}}_2)$ that satisfy the above two conditions, and divide this by the total number of pairs of configurations $(\overline{\boldsymbol{m}}_1, \overline{\boldsymbol{m}}_2)$. Let the number of such pairs of configurations be $N_{\mathsf{bad}}(n, \ell_0, \ell_1, \delta)$. Enumerating over all feasible values of m_1, m_2, ℓ_2, and using Corollary 3.15, we have

$$\begin{aligned} N_{\mathsf{bad}}(n, \ell_0, \ell_1, \delta) &\leq \sum_{m_1=\lfloor \ell_0(1-c/r) \rfloor + 1}^{\ell_0} |A_{n,\ell_0,\ell_1,m_1}| \cdot \left(\sum_{m_2=\lfloor \ell_1(1-c/r) \rfloor + 1}^{\ell_1} \sum_{\ell_2=m_2}^{\lceil \delta n \rceil - 1} |A_{n,\ell_1,\ell_2,m_2}| \right) \\ &\leq \sum_{m_1=\lfloor \ell_0(1-c/r) \rfloor + 1}^{\ell_0} \exp_2(g(n, \ell_0, \ell_1, m_1)) \cdot \left(\sum_{m_2=\lfloor \ell_1(1-c/r) \rfloor + 1}^{\ell_1} \sum_{\ell_2=m_2}^{\lceil \delta n \rceil - 1} \exp_2(g(n, \ell_1, \ell_2, m_2)) \right) , \end{aligned}$$

where g is defined in (5). The above sum can be upper bounded by its largest term multiplied by the number of terms.

- **Inner sum:** Firstly, consider any fixed m_1, m_2. Within the inner sum, we are interested in maximizing $\exp_2(g(n, \ell_1, \ell_2, m_2))$ over $\ell_2 \in [m_2, \lceil \delta n \rceil - 1]$. Since $g(n, \ell_1, \ell_2, m_2) = m_2 \cdot H\left(\frac{\ell_1 - m_2}{m_2}\right) + \ell_2 \cdot H\left(\frac{m_2}{\ell_2}\right) + (n - \ell_2) \cdot H\left(\frac{\ell_1 - m_2}{n - \ell_2}\right)$, and $\exp_2$ is monotonically increasing, this is equivalent to maximizing the following function:

$$f_{\mathsf{inner}}(\ell_2) := \ell_2 \cdot H\left(\frac{m_2}{\ell_2}\right) + (n - \ell_2) \cdot H\left(\frac{\ell_1 - m_2}{n - \ell_2}\right) \ ,$$

over $\ell_2 \in [m_2, \lceil \delta n \rceil - 1]$. We use the following claim.

Claim 3.17. Let n, ℓ_1, m_2 be fixed integers such that $m_2 < \ell_1$. Define

$$f_{\mathsf{inner}}(\ell_2) := \ell_2 \cdot H\left(\frac{m_2}{\ell_2}\right) + (n - \ell_2) \cdot H\left(\frac{\ell_1 - m_2}{n - \ell_2}\right) \ ,$$

Then, f_{inner} is increasing over $\ell_2 \in \left(m_2, \frac{m_2}{\ell_1} \cdot n\right)$.

Proof. See Section A. □

The largest value of ℓ_2 in the range of interest is $\lceil \delta n \rceil - 1$. Clearly,

$$\lceil \delta n \rceil - 1 < \delta n < (1 - c/r)n \leq \frac{m_2}{\ell_1} \cdot n \ ,$$

and therefore by Claim 3.17, each term in the inner sum can be upper bounded by

$$\exp_2\left(m_2 \cdot H\left(\frac{\ell_1 - m_2}{m_2}\right) + f_{\mathsf{inner}}(\delta n)\right)$$

Since there are $\leq (\delta n)$ terms in the inner sum, we can upper bound the inner sum by multiplying the above expression by (δn).

- **Middle sum:** Next, consider the middle sum over m_2. As noted in the previous step, each term of this sum can be upper bounded by

$$(\delta n) \cdot \exp_2\left(m_2 \cdot H\left(\frac{\ell_1 - m_2}{m_2}\right) + (\delta n) \cdot H\left(\frac{m_2}{\delta n}\right) + (n - \delta n) \cdot H\left(\frac{\ell_1 - m_2}{n - \delta n}\right)\right) \ .$$

We state the following claim.

Claim 3.18. Let $r \in \mathbb{N}$ be a repetition parameter and let c be a positive constant such that

$$c \leq \frac{r}{2} \cdot \left(1 - \frac{1}{\sqrt{5}}\right) \ . \tag{8}$$

Let $\delta \leq (1 - c/r)$, and n, ℓ_1 be fixed integers such that $\ell_1 < \delta n/(1 - c/r)$. Define

$$f_{\mathsf{middle}}(m_2) := m_2 \cdot H\left(\frac{\ell_1 - m_2}{m_2}\right) + (\delta n) \cdot H\left(\frac{m_2}{\delta n}\right) + (n - \delta n) \cdot H\left(\frac{\ell_1 - m_2}{n - \delta n}\right) \ .$$

Then, f_{middle} is decreasing over $m_2 \in (\ell_1(1 - c/r), \ell_1)$.

Proof. See Section A. □

The smallest value of m_2 in the range of interest is $\lfloor \ell_1(1-c/r) \rfloor + 1$. Clearly, $\lfloor \ell_1(1-c/r) \rfloor + 1 > \ell_1(1-c/r)$. Therefore, by Claim 3.18, each term in the middle sum can be upper bounded by

$$\begin{aligned} & (\delta n) \cdot \exp_2\left(f_{\mathsf{middle}}\left(\ell_1(1-c/r)\right)\right) \\ = \; & (\delta n) \cdot \exp_2\left(\ell_1(1-c/r) \cdot H\left(\frac{c}{r-c}\right) + (\delta n) \cdot H\left(\frac{\ell_1(1-c/r)}{\delta n}\right) + (n-\delta n) \cdot H\left(\frac{\ell_1 c/r}{n-\delta n}\right)\right) \end{aligned}$$

Since there are $\leq (\ell_1 c/r)$ terms in the middle sum, we can upper bound the middle sum by multiplying the above expression by $(\ell_1 c/r)$.

- **Outer sum:** Finally, consider the outer sum over m_1. Each term can be upper bounded by

$$(\ell_1 c/r) \cdot (\delta n) \cdot |A_{n,\ell_0,\ell_1,m_1}| \cdot \exp_2\left(f_{\mathsf{middle}}\left(\ell_1(1-c/r)\right)\right)$$

Only the term $|A_{n,\ell_0,\ell_1,m_1}|$ depends on m_1. By Corollary 3.15, $|A_{n,\ell_0,\ell_1,m_1}|$ can be upper bounded by $\exp_2(f_{\mathsf{outer}}(m_1))$, where

$$f_{\mathsf{outer}}(m_1) := m_1 \cdot H\left(\frac{\ell_0 - m_1}{m_1}\right) + \ell_1 \cdot H\left(\frac{m_1}{\ell_1}\right) + (n-\ell_1) \cdot H\left(\frac{\ell_0 - m_1}{n-\ell_1}\right) \quad .$$

Note the following two facts about ℓ_0, ℓ_1:

- $\ell_1 \leq \delta n/(1-c/r) \leq (1-c/r)n$ since $\delta \leq (1-c/r)^2$, and
- $\ell_0 \leq \ell_1/(1-c/r)$ since we condition on $\mathcal{F}_{\mathsf{mult1}}$ not occurring, so $\boldsymbol{w}_1$ must have at least $\ell_0(1-c/r)$ on-intervals, and each on-interval has length at least 1.

Given the above facts, if additionally (8) is satisfied, then we can use Claim 3.18 to show that $f_{\mathsf{outer}}(m_1)$ is non-increasing for $m_1 \geq \ell_0(1-c/r)$. Thus, every term in the outer sum can be upper bounded by

$$(\ell_1 c/r) \cdot (\delta n) \cdot \exp_2\left(f_{\mathsf{outer}}\left(\ell_0(1-c/r)\right) + f_{\mathsf{middle}}\left(\ell_1(1-c/r)\right)\right) \quad .$$

Since there are $\leq (\ell_0 c/r)$ terms in the outer sum, we can upper bound the outer sum by multiplying the above expression by $(\ell_0 c/r)$.

Thus, we can upper bound $N_{\mathsf{bad}}(n, \ell_0, \ell_1, \delta)$ as follows:

$$N_{\mathsf{bad}}(n,\ell_0,\ell_1,\delta) \leq (\ell_0 c/r) \cdot (\ell_1 c/r) \cdot (\delta n) \cdot \exp_2\left(f_{\mathsf{outer}}\left(\ell_0(1-c/r)\right) + f_{\mathsf{middle}}\left(\ell_1(1-c/r)\right)\right) \quad .$$

The total number of possible configuration pairs is $\binom{n}{\ell_0} \cdot \binom{n}{\ell_1}$. Therefore, we have

$$\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0,\ell_1,\overline{\boldsymbol{w}}_0,\delta) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) \leq N_{\mathsf{bad}}(\ell_0,\ell_1,\delta) \Big/ \left(\binom{n}{\ell_0} \cdot \binom{n}{\ell_1}\right)$$

By applying a union bound over all $\binom{n/r}{\ell_0/r}$ possible $\overline{\boldsymbol{w}}_0$, we get

$$\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0,\ell_1,\delta) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) \leq \binom{n/r}{\ell_0/r} \cdot N_{\mathsf{bad}}(\ell_0,\ell_1,\delta) \Big/ \left(\binom{n}{\ell_0} \cdot \binom{n}{\ell_1}\right)$$

$$\leq \frac{\ell_0\ell_1 c^2\delta n}{r^2} \cdot \frac{\exp_2\left(f_{\mathsf{outer}}\left(\ell_0(1-c/r)\right) + f_{\mathsf{middle}}\left(\ell_1(1-c/r)\right)\right) \cdot \binom{n/r}{\ell_0/r}}{\binom{n}{\ell_0}\cdot\binom{n}{\ell_1}}$$

$$\leq \frac{c^2 r\delta^3 n^3}{(r-c)^3} \cdot \frac{\exp_2\left(f_{\mathsf{outer}}\left(\ell_0(1-c/r)\right) + f_{\mathsf{middle}}\left(\ell_1(1-c/r)\right)\right) \cdot \binom{n/r}{\ell_0/r}}{\binom{n}{\ell_0}\cdot\binom{n}{\ell_1}}$$

$$\leq P_{\mathsf{perm}}(\delta)\cdot \exp_2(E) \ ,$$

where $P_{\mathsf{perm}}(\delta) := \frac{c^2 r\delta^3 n^3 (n+1)^2}{(r-c)^3}$ and E is the following exponent obtained by applying Fact 3.6 on the remaining binomial coefficients:

$$\begin{aligned} E &:= f_{\mathsf{outer}}\left(\ell_0(1-c/r)\right) + f_{\mathsf{middle}}\left(\ell_1(1-c/r)\right) + \frac{n}{r}\cdot H\left(\frac{\ell_0}{n}\right) - n\cdot H\left(\frac{\ell_0}{n}\right) - n\cdot H\left(\frac{\ell_1}{n}\right) \\ &= \left(\frac{1}{r}-1\right) n\cdot H\left(\frac{\ell_0}{n}\right) + \ell_0(1-c/r)\cdot H\left(\frac{c}{r-c}\right) + \ell_1\cdot H\left(\frac{\ell_0(1-c/r)}{\ell_1}\right) + (n-\ell_1)\cdot H\left(\frac{\ell_0 c}{r(n-\ell_1)}\right) \\ &\quad - n\cdot H\left(\frac{\ell_1}{n}\right) + \ell_1(1-c/r)\cdot H\left(\frac{c}{r-c}\right) + (\delta n)\cdot H\left(\frac{\ell_1(1-c/r)}{\delta n}\right) + (n-\delta n)\cdot H\left(\frac{\ell_1 c}{r(n-\delta n)}\right) \end{aligned}$$

Replacing $\ell_0 = d_0 n$ and $\ell_1 = d_1 n$ gives us the desired expression for $E_{\mathsf{perm}}(d_0, d_1, \delta)$ as defined in (7). □

3.5 Computing feasible δ

In Lemma 3.16, we have upper bounded $\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0,\ell_1,\delta) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}})$ for fixed $\ell_0 \in [\delta_{\mathcal{B}} n, \delta n/(1-c/r)^2]$ and $\ell_1 \in [\ell_0/(1-c/r), \delta n/(1-c/r)]$. The natural final step is to apply a union bound over all ℓ_0, ℓ_1 in these ranges in order to upper bound $\Pr(\mathcal{F}_{\mathsf{perm}}(\delta) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}})$, and then choose the largest value of δ that keeps the foregoing probability smaller than $2^{-\lambda-1}$. We only need to consider ℓ_0, ℓ_1 in these ranges since otherwise, by Lemmas 3.12 and 3.13, the probability of $\mathcal{F}_{\mathsf{perm}}(\ell_0,\ell_1,\delta)$ occurring is zero.

Unfortunately, we cannot analytically obtain a closed form solution for such a δ. This is because $E_{\mathsf{perm}}(d_0, d_1, \delta)$ in (7) is a transcendental function in δ since it involves both polynomial and logarithmic terms of δ. We also do not want to use simpler upper bounds as they would result in smaller feasible values of δ. Therefore, we present an algorithm CalcDist in Fig. 1 that numerically computes such a δ.

CalcDist $(\lambda, r, \delta_{\mathcal{B}}, n_{\min}, q_{\min})$:
Input parameters:
- Soundness parameter λ.
- Repetition parameter $r \geq 4$: larger values of r lead to better distance.
- Base code relative distance $\delta_{\mathcal{B}} \in (0,1)$: larger values of $\delta_{\mathcal{B}}$ lead to better distance.
- Minimum field size $q_{\min}$: larger values of $q_{\min}$ lead to better distance.
- Minimum block length $n_{\min}$: larger values of $n_{\min}$ lead to better distance.

Output: δ^* such that $\Pr(\mathcal{F}_{\mathsf{perm}}(\delta^*)) \leq 2^{-\lambda}$.
Algorithm:

1. Using $n_{\min}$ and $q_{\min}$, compute c by solving the following equation:

$$\left(\exp_2\left(\frac{r(2\lambda+4+4\log n_{\min})}{\delta_{\mathcal{B}} n_{\min}}\right)\cdot\left(\left(\frac{1}{\delta_{\mathcal{B}}}\right)\cdot e^{(r-r\alpha+1+c\alpha)}\cdot\frac{\left(\frac{r}{c}\right)^{c\alpha}}{(1-\alpha)^{(r-r\alpha)}}\right)\right)^{\frac{1}{c-1}} = q_{\min}-1 \ .$$

Check that $1 < c < r/2 \cdot (1 - 1/\sqrt{5})$; if not, return FAIL.

2. Using $n_{\min}$ and computed value of c, compute γ:

$$\gamma = \left(\lambda + 1 + \log_2\left(\frac{c^2(r-c)^3}{r^5}\right) + 7\log(n_{\min}+1)\right) \Big/ n_{\min} \ .$$

3. Compute d_1^* and δ_1^* that satisfy the following two equations:

$$\begin{aligned} \frac{\partial}{\partial d_1}\left(E_{\mathsf{perm}}(\delta_{\mathcal{B}}, d_1, \delta_1^*)/n\right)\Big|_{d_1=d_1^*} &= 0 \ , \\ \gamma + E_{\mathsf{perm}}(\delta_{\mathcal{B}}, d_1^*, \delta_1^*)/n &= 0 \ , \end{aligned}$$

where $E_{\mathsf{perm}}(\cdot,\cdot,\cdot)$ is the function defined in (7). Use numerical methods (e.g., Newton–Raphson method) to solve the above system of equations. If no solution is found, return FAIL.

4. Compute d_0^*, d_1^*, δ_2^* that satisfy the following three equations:

$$\begin{aligned} \frac{\partial}{\partial d_0}\left(E_{\mathsf{perm}}(d_0, d_1, \delta_2^*)/n\right)\Big|_{d_0=d_0^*, d_1=d_1^*} &= 0 \ , \\ \frac{\partial}{\partial d_1}\left(E_{\mathsf{perm}}(d_0, d_1, \delta_2^*)/n\right)\Big|_{d_0=d_0^*, d_1=d_1^*} &= 0 \ , \\ \det\left(\boldsymbol{H}_{\delta_2^*}(d_0, d_1)\right)\Big|_{d_0=d_0^*, d_1=d_1^*} &= 0 \ , \end{aligned} \tag{9}$$

where $\boldsymbol{H}_{\delta_2^*}(d_0, d_1)$ is the Hessian of $E_{\mathsf{perm}}(d_0, d_1, \delta_2^*)/n$ with respect to (d_0, d_1). If multiple solutions are found, pick the smallest value of δ_2^*.

5. Return $\delta^* = \min(\delta_1^*, \delta_2^*)$.

Figure 1: CalcDist: outputs δ^* such that $\Pr(\mathcal{F}_{\mathsf{perm}}(\delta^*)) \le 2^{-\lambda}$.

We now prove the correctness of CalcDist.

Lemma 3.19. *Given inputs $(\lambda, r, \delta_{\mathcal{B}}, n, q)$ such that the conditions in Fig. 1 are satisfied, let δ^* be the output of* CalcDist. *Then, we have*

$$\Pr(\mathcal{F}_{\mathsf{perm}}(\delta^*) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) \le 2^{-\lambda-1} \ .$$

Consequently, since n and q satisfy (1) and (2) respectively, by Lemmas 3.8 and 3.11, $\Pr(\mathcal{F}_{\mathsf{perm}}(\delta^*)) \le 2^{-\lambda}$.

Proof. Define $A := (1 - c/r)$, $B := c/r$, $d_0 := \ell_0/n$, and $d_1 := \ell_1/n$.

We first discuss the trivial cases. If $\ell_0 \ge \lfloor \delta^* n/A^2 \rfloor$, then by Lemma 3.12 we have $\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta^*) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) = 0$, and the claim holds trivially. Similarly, if $\ell_1 \ge \lfloor \delta^* n/A \rfloor$, then by Lemma 3.13 we have $\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta^*) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) = 0$, and the claim holds trivially. Thus, the non-trivial case is when $\ell_0 < \lfloor \delta^* n/A^2 \rfloor$ and $\ell_1 < \lfloor \delta^* n/A \rfloor$. Observe that if we prove that $\Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta^*) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) \le 2^{-\lambda-1} \cdot n^{-2}$ for all $\ell_0 \in [\delta_{\mathcal{B}} n, n]$ and $\ell_1 \in [\ell_0 A, n]$, then applying the union bound

over all such ℓ_0, ℓ_1 gives the desired result. Therefore, say we have proven the following claim about the δ^*, the output of CalcDist in Fig. 1:

$$E_{\mathsf{perm}}(d_0, d_1, \delta^*) \le -\gamma n \text{ for all } d_0 \in [\delta_{\mathcal{B}}, \delta^*/A^2] \text{ and } d_1 \in [\delta_{\mathcal{B}} A, \delta^*/A] \ . \tag{10}$$

Then, we can achieve our desired result by using Lemma 3.16:

$$\begin{aligned}
& \Pr(\mathcal{F}_{\mathsf{perm}}(\ell_0, \ell_1, \delta^*) \mid \neg\mathcal{F}_{\mathsf{mult1}}, \neg\mathcal{F}_{\mathsf{mult2}}) \\
\le\ & P_{\mathsf{perm}}(\delta^*) \cdot \exp_2(E_{\mathsf{perm}}(d_0, d_1, \delta^*)) \\
=\ & P_{\mathsf{perm}}(\delta^*) \cdot \exp_2(-\gamma n) && \text{[by definition of } \delta^* \text{ in Fig. 1]} \\
\le\ & \frac{c^2(r-c)^3}{r^5} \cdot (n+1)^5 \cdot \exp_2(-\gamma n) && \text{[by (6) and } \delta^* < (1-c/r)^2\text{]} \\
\le\ & 2^{-\lambda-1} \cdot n^{-2} \ . && \text{[by (1)]}
\end{aligned}$$

Thus, the challenge lies in proving (10). Intuitively, $E_{\mathsf{perm}}(d_0, d_1, \delta^*)$ should be maximized at $d_0 = \delta_{\mathcal{B}}$ and $d_1 = \delta_{\mathcal{B}} A$: the lower the inputs weights, the higher should be the probability of failure. If this were true, we can simply return the largest value of δ that satisfies $E_{\mathsf{perm}}(\delta_{\mathcal{B}}, \delta_{\mathcal{B}} A, \delta) \le -\gamma n$. Unfortunately, this is not necessarily true, so we have to find the maximizing values of d_0 and d_1 (at least implicitly). To this end, we state the following claims, which are all proven in Section A.

Claim 3.20. Consider any fixed $\delta < A^2$ and $d_1 \in [\delta_{\mathcal{B}} A, \delta/A)$. Then,

$$\frac{\partial^2}{\partial d_0^2} E_{\mathsf{perm}}(d_0, d_1, \delta) < 0 \text{ for all } d_0 \in (0, d_1/A) \ .$$

That is, $E_{\mathsf{perm}}(d_0, d_1, \delta)$ is concave in d_0 in the specified range.

Claim 3.21. Consider any fixed $\delta < A^2$ and $d_1 \in [\delta_{\mathcal{B}} A, \delta/A)$. Then, there exists a unique value $d_0^*(d_1) \in (0, d_1/A)$ such that

$$\frac{\partial}{\partial d_0} E_{\mathsf{perm}}(d_0, d_1, \delta)\Big|_{d_0 = d_0^*(d_1)} = 0 \ .$$

Note that $d_0^*(d_1)$ is a function of d_1, but not of δ. Furthermore, $d_0^*(d_1)$ is increasing in d_1.

Claim 3.22. Consider any fixed $d_0 \in [\delta_{\mathcal{B}}, \delta/A^2)$ and $d_1 \in [d_0 A, \delta/A)$. Then, we have

$$\frac{\partial}{\partial \delta} E_{\mathsf{perm}}(d_0, d_1, \delta) > 0 \ .$$

That is, $E_{\mathsf{perm}}(d_0, d_1, \delta)$ is increasing in δ in the specified ranges of d_0, d_1, for any feasible value of δ.

Claim 3.23. Define the following function

$$g(d_1, \delta) := \sup_{d_0 \in (0, d_1/A)} E_{\mathsf{perm}}(d_0, d_1, \delta)/n \ . \tag{11}$$

Then, if $\delta < A$, we have

$$\frac{\partial}{\partial \delta} \frac{\partial}{\partial d_1} g(d_1, \delta) > 0 \text{ for all } d_1 \in [\delta_{\mathcal{B}} A, \delta/A) \ .$$

That is, $\frac{\partial}{\partial d_1} g(d_1, \delta)$ is increasing in δ in the specified range of d_1, for any feasible value of δ.

Claim 3.24. If $r \geq 4$ and $c < r/2$, there exists some sufficiently small $\delta_0 > 0$ such that

$$\frac{\partial}{\partial d_1} g(d_1, \delta_0) < 0 \text{ for all } d_1 \in [\delta_{\mathcal{B}} A, \delta_0/A) \ ,$$

where $g(d_1, \delta)$ is defined in (11).

Assuming these claims hold, we can prove (10) as follows. By Claim 3.21, for any fixed $\delta < A^2$ and $d_1 \in [\delta_{\mathcal{B}} A, \delta/A)$, there exists a unique value $d_0^*(d_1)$ such that

$$\frac{\partial}{\partial d_0} E_{\mathsf{perm}}(d_0, d_1, \delta)\Big|_{d_0 = d_0^*(d_1)} = 0 \ .$$

By Claim 3.20, it must be that $E_{\mathsf{perm}}(d_0, d_1, \delta)$ is maximized at $d_0 = d_0^*(d_1)$ for any fixed d_1, δ in the specified ranges. Therefore, we have

$$g(d_1, \delta) = E_{\mathsf{perm}}(d_0^*(d_1), d_1, \delta)/n \ ,$$

where $g(d_1, \delta)$ is defined in (11).

Now, say we want to prove (10) for some fixed $d_1 \in (\delta_{\mathcal{B}} A, \delta^*/A)$. There are two cases to consider.

- $d_0^*(d_1) < \delta_{\mathcal{B}}$. In this case, $E_{\mathsf{perm}}(d_0, d_1, \delta^*)$ is maximized at $d_0 = d_0^*(d_1) < \delta_{\mathcal{B}}$. By Claim 3.20, $E_{\mathsf{perm}}(d_0, d_1, \delta^*)$ is decreasing in d_0 for $d_0 > d_0^*(d_1)$. Since $\delta_{\mathcal{B}} > d_0^*(d_1)$, the maximum value of $E_{\mathsf{perm}}(d_0, d_1, \delta^*)$ for $d_0 \in [\delta_{\mathcal{B}}, \delta^*/A^2]$ is attained at $d_0 = \delta_{\mathcal{B}}$. We have

$$\begin{aligned} E_{\mathsf{perm}}(\delta_{\mathcal{B}}, d_1, \delta^*) &\leq E_{\mathsf{perm}}(\delta_{\mathcal{B}}, d_1, \delta_1^*) && \text{[By Claim 3.22]} \\ &\leq E_{\mathsf{perm}}(\delta_{\mathcal{B}}, d_1^*, \delta_1^*) && [d_1^* \text{ as defined in Step 3 in Fig. 1}] \\ &\leq -\gamma n && \text{[By definition of } \delta_1^* \text{ in Fig. 1]} \end{aligned}$$

- $d_0^*(d_1) \geq \delta_{\mathcal{B}}$. Firstly, let $(d_0^*)^{-1}(\cdot)$ be the inverse function of $d_0^*(\cdot)$. This inverse exists and is increasing in d_0 as a consequence of Claim 3.21, which states that $d_0^*(d_1)$ is increasing in d_1. Since we want to show that $E_{\mathsf{perm}}(d_0^*(d_1), d_1, \delta^*) \leq \gamma n$ or $g(d_1, \delta^*) \leq -\gamma$ for any fixed d_1 such that $d_0^*(d_1) \geq \delta_{\mathcal{B}}$, it suffices to prove the following two statements:

  – $g(d_1, \delta^*) \leq -\gamma$ at the smallest such d_1, i.e., at $d_1 = (d_0^*)^{-1}(\delta_{\mathcal{B}})$.

  – $g(d_1, \delta^*)$ is non-increasing in d_1 over the range $[(d_0^*)^{-1}(\delta_{\mathcal{B}}), \delta^*/A)$.

  The first statement follows from the following:

$$\begin{aligned} g(d_1, \delta^*)\Big|_{d_1 = (d_0^*)^{-1}(\delta_{\mathcal{B}})} &= E_{\mathsf{perm}}(\delta_{\mathcal{B}}, (d_0^*)^{-1}(\delta_{\mathcal{B}}), \delta^*)/n \\ &\leq E_{\mathsf{perm}}(\delta_{\mathcal{B}}, (d_0^*)^{-1}(\delta_{\mathcal{B}}), \delta_1^*)/n && \text{[By Claim 3.22]} \\ &\leq E_{\mathsf{perm}}(\delta_{\mathcal{B}}, d_1^*, \delta_1^*)/n && [d_1^* \text{ as defined in Step 3 in Fig. 1}] \\ &\leq -\gamma && \text{[By definition of } \delta_1^* \text{ in Fig. 1]} \end{aligned}$$

  Thus, all that remains is to prove the second statement, and we must find the largest value of δ such that

$$\frac{\partial}{\partial d_1} g(d_1, \delta) \leq 0 \text{ for all } d_1 \in [(d_0^*)^{-1}(\delta_{\mathcal{B}}), \delta/A) \ . \tag{12}$$

Now, we define the following:

$$h(\delta) := \sup_{d_1 \in [(d_0^*)^{-1}(\delta_{\mathcal{B}}), \delta/A)} \frac{\partial}{\partial d_1} g(d_1, \delta) \ .$$

By Claim 3.23, $\frac{\partial}{\partial d_1} g(d_1, \delta)$ is increasing in δ for any fixed d_1 in the specified range. Therefore $h(\delta)$ is also increasing in δ. Since $\lim_{\delta \to A^2} h(\delta) > 0$, h is continuous, and by Claim 3.24 there exists δ_0 such that $h(\delta_0) < 0$, the intermediate value theorem implies that there exists some $\delta' < A^2$ such that $h(\delta') = 0$. Thus, there exists a value of $d_1' \in [(d_0^*)^{-1}(\delta_{\mathcal{B}}), \delta'/A)$ and δ' such that

$$\frac{\partial}{\partial d_1} g(d_1, \delta')\Big|_{d_1 = d_1'} = 0 \ , \tag{13}$$

$$\frac{\partial^2}{\partial d_1^2} g(d_1, \delta')\Big|_{d_1 = d_1'} = 0 \ , \tag{14}$$

and for all $d_1 \in [(d_0^*)^{-1}(\delta_{\mathcal{B}}), \delta'/A)$, we have $\frac{\partial}{\partial d_1} g(d_1, \delta') \leq 0$.

Expanding the left-hand side of (13) via the multivariate chain rule, we get

$$\begin{aligned}
&\frac{\partial}{\partial d_1} E_{\mathsf{perm}}(d_0^*(d_1), d_1, \delta')\Big|_{d_1 = d_1'} \\
&= \frac{\partial}{\partial d_1} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0 = d_0^*(d_1'), d_1 = d_1'} + \frac{\partial}{\partial d_0} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0 = d_0^*(d_1'), d_1 = d_1'} \cdot \frac{\partial}{\partial d_1} d_0^*(d_1)\Big|_{d_1 = d_1'} \\
&= \frac{\partial}{\partial d_1} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0 = d_0^*(d_1'), d_1 = d_1'} \ ,
\end{aligned}$$

where the last step follows from the fact that $\frac{\partial}{\partial d_0} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0 = d_0^*(d_1'), d_1 = d_1'} = 0$ by definition of $d_0^*(d_1')$. Thus, (13) is equivalent to

$$\frac{\partial}{\partial d_1} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0 = d_0^*(d_1'), d_1 = d_1'} = 0 \ . \tag{15}$$

Similarly, expanding the left-hand side of (14) via the multivariate chain rule, we get

$$\begin{aligned}
&\frac{\partial^2}{\partial d_1^2} E_{\mathsf{perm}}(d_0^*(d_1), d_1, \delta')\Big|_{d_1 = d_1'} \\
&= \frac{\partial}{\partial d_1} \left(\frac{\partial}{\partial d_1} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0 = d_0^*(d_1)} \right) \Big|_{d_1 = d_1'} \\
&= \frac{\partial^2}{\partial d_1^2} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0 = d_0^*(d_1'), d_1 = d_1'} + \frac{\partial^2}{\partial d_0 \partial d_1} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0 = d_0^*(d_1'), d_1 = d_1'} \cdot \frac{\partial}{\partial d_1} d_0^*(d_1)\Big|_{d_1 = d_1'}
\end{aligned}$$

In this case, we need to compute the value of $\frac{\partial}{\partial d_1} d_0^*(d_1)$ to further simplify the expression. Since $d_0^*(d_1)$ is defined as the unique root of $\frac{\partial}{\partial d_0} E_{\mathsf{perm}}(d_0, d_1, \delta') = 0$, we can implicitly differentiate this equation with respect to d_1 to get

$$\frac{\partial^2}{\partial d_0 \partial d_1} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0 = d_0^*(d_1), d_1 = d_1} + \frac{\partial^2}{\partial d_0^2} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0 = d_0^*(d_1), d_1 = d_1} \cdot \frac{\partial}{\partial d_1} d_0^*(d_1) = 0 \ .$$

Rearranging and solving for $\frac{\partial}{\partial d_1} d_0^*(d_1)$, and then substituting into the previous expression, we get

$$\begin{aligned}
&\frac{\partial^2}{\partial d_1^2} E_{\mathsf{perm}}(d_0^*(d_1), d_1, \delta')\Big|_{d_1=d_1'} \\
&= \frac{\partial^2}{\partial d_1^2} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0=d_0^*(d_1'),d_1=d_1'} - \frac{\left(\frac{\partial^2}{\partial d_0 \partial d_1} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0=d_0^*(d_1'),d_1=d_1'}\right)^2}{\frac{\partial^2}{\partial d_0^2} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0=d_0^*(d_1'),d_1=d_1'}} \\
&= \frac{\det(\boldsymbol{H}_{\delta'}(d_0, d_1))}{\frac{\partial^2}{\partial d_0^2} E_{\mathsf{perm}}(d_0, d_1, \delta')}\Big|_{d_0=d_0^*(d_1'),d_1=d_1'} ,
\end{aligned}$$

where $\boldsymbol{H}_{\delta'}(d_0, d_1)$ is the Hessian of $E_{\mathsf{perm}}(d_0, d_1, \delta')$ with respect to d_0 and d_1. Claim 3.20 implies that $\frac{\partial^2}{\partial d_0^2} E_{\mathsf{perm}}(d_0, d_1, \delta')\Big|_{d_0=d_0^*(d_1'),d_1=d_1'} < 0$. Thus, (14) is equivalent to

$$\det(\boldsymbol{H}_{\delta'}(d_0, d_1))\Big|_{d_0=d_0^*(d_1'),d_1=d_1'} = 0 \ . \tag{16}$$

To summarize, the largest value of δ that satisfies (12) must satisfy (15) and (16) simultaneously where the value of d_0 satisfies $\frac{\partial}{\partial d_0} E_{\mathsf{perm}}(d_0, d_1, \delta) = 0$. Thus, given that there exists δ_0 such that (12) is satisfied (by Claim 3.24), there exists a δ' that satisfies the system of equations defined in Step 4 of Fig. 1, and additionally satisfies (12). Since Fig. 1 computes the smallest value of δ_2^* that satisfies the system of equations defined in Step 4, we have $\delta_2^* \leq \delta'$. The final returned value is $\delta^* = \min(\delta_1^*, \delta_2^*) \leq \delta_2^* \leq \delta'$. Since δ' satisfies (12), by Claim 3.23, δ^* also satisfies (12), as desired.

This concludes the proof of (10), and therefore the lemma. □

Remark 3.25 (Need for δ_2^*)**.** In order to simplify the algorithm CalcDist in Fig. 1, one may consider only computing δ_1^* as the output, i.e., returning the largest value of δ such that $E_{\mathsf{perm}}(\delta_{\mathcal{B}}, \delta_{\mathcal{B}} A, \delta) \leq -\gamma n$. In practice for $n = 2^{20}$ and $q = 2^{256}$, we observe that δ_1^* is often smaller than δ_2^*, except when $r \in \{4, 5\}$, in which case δ_2^* is smaller. Thus, it is the small r case that necessitates the computation of δ_2^*. We highly recommend that the reader experiment with the Jupyter notebook provided in this repository (https://anonymous.4open.science/r/era-code-analysis-1DFF) to see this phenomenon for themselves.

4 IOPP preliminaries

In this section, we present the preliminary definitions and notation used throughout our IOPP construction.

4.1 Notation

For a finite field $\mathbb{F}$ and $n \in \mathbb{N}$, bold lowercase letters $\boldsymbol{x} \in \mathbb{F}^n$ denote vectors of field elements. Unless otherwise specified, we assume that vectors over $\mathbb{F}$ are zero-indexed. $\boldsymbol{x}[i]$ denotes the i-th element of $\boldsymbol{x}$.[10] Given $\boldsymbol{x} \in \mathbb{F}^n$ and $y \in \mathbb{F}$, we use $y \cdot \boldsymbol{x}$ to denote $[y \cdot \boldsymbol{x}[i]]_{i \in [n]}$. Given $\ell \in \mathbb{N}$, we use $\boldsymbol{x}^\ell$ to denote the vector $(\boldsymbol{x}, \ldots, \boldsymbol{x})$, where $\boldsymbol{x}$ is repeated ℓ times. Given a set $S \subseteq [n]$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^n$, we say $\boldsymbol{x}[S] = \boldsymbol{y}[S]$ if $\boldsymbol{x}[i] = \boldsymbol{y}[i]$ for all $i \in S$.

We use bold uppercase letters $\boldsymbol{M} \in \mathbb{F}^{n \times m}$ to denote matrices over $\mathbb{F}$. $\boldsymbol{M}[i, j]$, $\boldsymbol{M}[i, \cdot]$, and $\boldsymbol{M}[\cdot, j]$ denote the element in the i-th row and j-th column, the i-th row, and the j-th column of $\boldsymbol{M}$ respectively.

Binary representation. Given $i, n \in \mathbb{N}$, $\langle i \rangle_n$ represents the n-bit binary representation of i in most-significant-bit-first order. We drop the subscript n when it is clear from context. We also define the inverse operation $\mathsf{int}(\boldsymbol{x})$ which maps $\boldsymbol{x} = \langle v \rangle_n \in \{0,1\}^n$ to v.

Inner products. Given $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^n$, we use $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ to denote the inner product $\sum_{i=1}^n \boldsymbol{x}[i] \cdot \boldsymbol{y}[i]$, and we use $\langle \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \rangle$ to denote the 'triple inner product' $\sum_{i=1}^n \boldsymbol{a}[i] \cdot \boldsymbol{b}[i] \cdot \boldsymbol{c}[i]$.

Multilinear polynomials. We use $\mathbb{F}^{\leq d}[X_1, \ldots, X_\mu]$ to denote the set of all μ-variate polynomials over $\mathbb{F}$ such that the degree of each variable in every monomial is at most d. We denote a multivariate polynomial by $p(X_1, \ldots, X_\mu)$ or by $p(\boldsymbol{X})$ when μ is clear from context. $\mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ is the set of all μ-variate *multilinear* polynomials. $p(\boldsymbol{X}) \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ is uniquely defined by its evaluations $\boldsymbol{p} := [p(\langle i \rangle)]_{i=0}^{2^\mu - 1}$ over the μ-dimensional boolean hypercube $\{0,1\}^\mu \subset \mathbb{F}^\mu$.

Multilinear extension. The *multilinear extension* of a vector $\boldsymbol{m} \in \mathbb{F}^{2^\mu}$ is the unique multilinear polynomial $\hat{m}(\boldsymbol{X}) \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ such that $\hat{m}(\langle i - 1 \rangle) = \boldsymbol{m}[i]$ for all $i \in [2^\mu]$.

Lagrange basis polynomial. The *Lagrange basis polynomial* $\mathrm{eq}_\mu(\boldsymbol{X}, \boldsymbol{Y}) := \prod_{i=1}^\mu (X_i Y_i + (1 - X_i)(1 - Y_i))$ is a multilinear polynomial that evaluates to 1 if and only if $\boldsymbol{X} = \boldsymbol{Y}$ for any $\boldsymbol{X}, \boldsymbol{Y} \in \{0,1\}^\mu$. We omit μ when it is clear from context. For any $p(\boldsymbol{X}) \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ and $\boldsymbol{z} \in \mathbb{F}^\mu$, we have

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

We use $\mathbf{eq}(\boldsymbol{z})$ to denote the vector of evaluations of $\mathrm{eq}(\boldsymbol{X}, \boldsymbol{z})$ over $\{0,1\}^\mu$.

Hamming distance. We use Δ to denote total Hamming distance, i.e., for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^n$, $\Delta(\boldsymbol{x}, \boldsymbol{y}) := |\{i \in [n] : \boldsymbol{x}[i] \neq \boldsymbol{y}[i]\}|$, and δ to denote fractional Hamming distance, i.e., $\delta(\boldsymbol{x}, \boldsymbol{y}) := \Delta(\boldsymbol{x}, \boldsymbol{y})/n$. For a set $S \subset \mathbb{F}^n$ and an element $\boldsymbol{x} \in \mathbb{F}^n$, we define $\Delta(\boldsymbol{x}, S) := \min_{\boldsymbol{s} \in S} \Delta(\boldsymbol{x}, \boldsymbol{s})$, and $\delta(\boldsymbol{x}, S) := \Delta(\boldsymbol{x}, S)/n$. We also use $\boldsymbol{x} \sim_\varepsilon \boldsymbol{y}$ to denote $\delta(\boldsymbol{x}, \boldsymbol{y}) < \varepsilon$.

4.2 Coding theory

Definition 4.1 (Linear codes)**.** A linear code $\mathcal{C}$ of message length k, block length n, over an alphabet $\mathbb{F}$ is a k-dimensional subspace of $\mathbb{F}^n$. The *rate* of the code is defined to be k/n and the *distance* of a code $\mathcal{C}$ is defined to be

$$\delta_{\mathcal{C}} := \min_{\boldsymbol{x} \neq \boldsymbol{y} \in \mathcal{C}} \delta(\boldsymbol{x}, \boldsymbol{y})$$

[10] When it helps to make notation more clear and does not lead to ambiguity, we sometimes use x_i to denote the same.

We abbreviate the parameters of a code as $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$. We use $\boldsymbol{c} \in \mathcal{C}$ to denote codewords and $\boldsymbol{w} \in \mathbb{F}^n$ to denote arbitrary words. We say that a word $\boldsymbol{w} \in \mathbb{F}^n$ is δ-far from a code $\mathcal{C}$ if $\delta(\boldsymbol{w}, \mathcal{C}) > \delta$, and δ-close to $\mathcal{C}$ if $\delta(\boldsymbol{w}, \mathcal{C}) \leq \delta$.

Encoding. An *encoding function* of a code $\mathcal{C}$ is a bijection $\mathsf{Enc}_{\mathcal{C}} : \mathbb{F}^k \to \mathcal{C}$. The message $\boldsymbol{m} \in \mathbb{F}^k$ of a codeword $\boldsymbol{c} \in \mathcal{C}$ relative to an encoding function $\mathsf{Enc}_{\mathcal{C}}$ is the unique vector such that $\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}) = \boldsymbol{c}$. If $\mathcal{C}$ is linear, there exists a $n \times k$ matrix $\boldsymbol{G}_{\mathcal{C}}$ known as the *generator matrix* such that $\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}) = \boldsymbol{G}_{\mathcal{C}}\boldsymbol{m}$. We also use $T_{\mathcal{C}}$ to denote the time complexity of encoding a message under $\mathcal{C}$.

Definition 4.2 (Interleaved codes)**.** Let $\mathcal{B}$ be a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{B}})$. Given an *interleaving parameter* $t \in \mathbb{N}$, the t-interleaved code $\mathcal{B}^t \subseteq (\mathbb{F}^t)^n$ is a linear code with parameters $(\mathbb{F}^t, n, k, \delta_{\mathcal{B}})$ such that for every $\boldsymbol{C} \in \mathcal{B}^t$, every row of $\boldsymbol{C}$ is a codeword of $\mathcal{B}$, when $\boldsymbol{C}$ is interpreted as a matrix in $\mathbb{F}^{t \times n}$.

Given a message $\boldsymbol{M} \in \mathbb{F}^{t \times k}$, $\mathsf{Enc}_{\mathcal{B}^t}(\boldsymbol{M})$ is obtained by encoding each row of $\boldsymbol{M}$ under $\mathcal{B}$. Importantly, the distance of $\mathcal{B}^t$ is the same as that of $\mathcal{B}$ because we define the interleaved code over the larger alphabet $\mathbb{F}^t$.

Definition 4.3 (Tensor codes)**.** Let $\mathbb{F}$ be a finite field, $\mathcal{C}_1$ be a linear code with parameters $(\mathbb{F}, n_1, k_1, \delta_{\mathcal{C}_1})$ and generator matrix $\boldsymbol{G}_{\mathcal{C}_1} \in \mathbb{F}^{n_1 \times k_1}$, and $\mathcal{C}_2$ be a linear code with parameters $(\mathbb{F}, n_2, k_2, \delta_{\mathcal{C}_2})$ and generator matrix $\boldsymbol{G}_{\mathcal{C}_2} \in \mathbb{F}^{n_2 \times k_2}$. The *tensor code* $\mathcal{C}_1 \otimes \mathcal{C}_2$ is the linear code with the generator matrix $\boldsymbol{G}_{\mathcal{C}_1} \otimes \boldsymbol{G}_{\mathcal{C}_2}$, where $\otimes$ denotes the Kronecker product. It is well known that $\mathcal{C}_1 \otimes \mathcal{C}_2$ has parameters $(\mathbb{F}, n_1 n_2, k_1 k_2, \delta_{\mathcal{C}_1}\delta_{\mathcal{C}_2})$. The encoding time of $\mathcal{C}_1 \otimes \mathcal{C}_2$ is $O(n_2 \cdot T_{\mathcal{C}_1} + k_1 \cdot T_{\mathcal{C}_2})$ (first encoding the rows under $\mathcal{C}_2$, then encoding the columns under $\mathcal{C}_1$).

Definition 4.4 (List decoding)**.** Let $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$, let $\boldsymbol{w} \in \mathbb{F}^n$ be a word, and let δ be a proximity parameter. We define

$$\Lambda_{\mathcal{C}}(\boldsymbol{w}, \delta) := \{\boldsymbol{c} \in \mathcal{C} : \delta(\boldsymbol{c}, \boldsymbol{w}) \leq \delta\} \quad .$$

We also use $\Lambda(\mathcal{C}, \delta)$ to denote $\Lambda_{\mathcal{C}}(\boldsymbol{w}, \delta)$ for some word $\boldsymbol{w}$ such that $|\Lambda_{\mathcal{C}}(\boldsymbol{w}, \delta)|$ is maximized. We say that $\mathcal{C}$ is (δ, ℓ)-*list decodable* if $|\Lambda(\mathcal{C}, \delta)| \leq \ell$. We say that δ is in the 'unique decoding regime' if $\mathcal{C}$ is $(\delta, 1)$-list decodable, and in the 'list decoding regime' if $\mathcal{C}$ is (δ, ℓ)-list decodable for some $\ell = \mathrm{poly}(n)$.

Mutual Correlated Agreement. Linear codes exhibit a property called *correlated agreement*, which is used extensively in previous work on IOPPs and accumulation schemes [AHIV17; BBHR18; ACFY24]. A stronger notion known as *mutual correlated agreement* (MCA) was introduced in [ACFY25], but has been explored only for Reed–Solomon codes. Related but slightly 'weaker' notions that apply to arbitrary linear codes in the list decoding regime were introduced in [GKL24; Zei24].

Definition 4.5 (Mutual correlated agreement for proximity generators)**.** Let $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$. We say that Gen is a proximity generator for $\mathcal{C}$ with mutual correlated agreement with proximity bound B^* and error $\varepsilon_{\mathsf{mca}}$ if for every $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_\ell \in \mathbb{F}^n$ and $\delta \in (0, 1 - \mathsf{B}^*(\mathcal{C}, \ell))$ the following holds:

$$\Pr_{(r_1,\ldots,r_\ell) \leftarrow \mathsf{Gen}(\ell)} \left[\exists S \subset [n] \quad \text{s.t.} \quad \begin{array}{l} |S| \geq (1-\delta)n \ , \\ \exists \boldsymbol{c} \in \mathcal{C} : \sum_{j \in [\ell]} r_j \cdot \boldsymbol{w}_j[S] = \boldsymbol{c}[S] \ , \\ \exists i \in [\ell], \forall \boldsymbol{c}' \in \mathcal{C} : \boldsymbol{w}_i[S] \neq \boldsymbol{c}'[S] \end{array} \right] \leq \varepsilon_{\mathsf{mca}}(\mathcal{C}, \ell, \delta) \ .$$

In words, mutual correlated agreement states that with probability at most $\varepsilon_{\mathsf{mca}}(\mathcal{C}, \ell, \delta)$, a linear combination of words $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_\ell$ with respect to coefficients $r_1, \ldots, r_\ell$ can only agree with a codeword $\boldsymbol{c}$ on a large set S if each $\boldsymbol{w}_j$ agrees with some codeword $\boldsymbol{c}_j$ on S. Improving mutual correlated agreement bounds is an active area of research, and a recent survey covers state-of-the-art results for a variety of codes [ABF26].

4.3 Useful results regarding polynomials

Lemma 4.6 (Ore-DeMillo-Lipton-Schwartz-Zippel lemma)**.** *Let $f \in \mathbb{F}^{\leq d}[X_1, \ldots, X_\mu]$ be a non-zero polynomial. Then*

$$\Pr_{\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^\mu} [f(\boldsymbol{r}) = 0] \leq \tfrac{d \cdot \mu}{|\mathbb{F}|} \ .$$

Lemma 4.7 (Out-of-domain evaluation)**.** *Let $\mathcal{C}$ be a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$, let $\boldsymbol{w} \in \mathbb{F}^n$ be an arbitrary word, and let $\delta \in (0, \delta_{\mathcal{C}})$. Then, we have*

$$\Pr_{\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^{\log k}} \left[\exists \textit{ distinct } \boldsymbol{u}, \boldsymbol{v} : \mathbb{F}^k \textit{ s.t. } \begin{array}{r} \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{u}) \in \Lambda_{\mathcal{C}}(\boldsymbol{w}, \delta) \\ \textit{and } \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{v}) \in \Lambda_{\mathcal{C}}(\boldsymbol{w}, \delta) \\ \textit{and } \hat{u}(\boldsymbol{r}) = \hat{v}(\boldsymbol{r}) \end{array} \right] \leq \frac{|\Lambda(\mathcal{C}, \delta)|^2}{2} \cdot \frac{\log k}{|\mathbb{F}|} \ .$$

Proof. If $\hat{u}(\boldsymbol{X}) \neq \hat{v}(\boldsymbol{X})$, then $\hat{u}(\boldsymbol{r}) = \hat{v}(\boldsymbol{r})$ with probability at most $\log k / |\mathbb{F}|$ by Lemma 4.6. Applying the union bound over all pairs $(\boldsymbol{u}, \boldsymbol{v})$ such that $\mathsf{Enc}_{\mathcal{C}}(\boldsymbol{u}), \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{v}) \in \Lambda_{\mathcal{C}}(\boldsymbol{w}, \delta)$, we get the desired bound. □

The sumcheck protocol. Let $p(\boldsymbol{X})$ be a μ-variate polynomial of degree at most d over a field $\mathbb{F}$. The seminal sumcheck protocol of Lund, Fortnow, Karloff, and Nisan [LFKN92] reduces the problem of checking that $\sum_{\boldsymbol{b} \in \{0,1\}^\mu} p(\boldsymbol{b}) = \sigma$ to checking that $p(\boldsymbol{r}) = y_{\boldsymbol{r}}$, where $\boldsymbol{r} \in \mathbb{F}^\mu$ consists of random challenges sampled by the verifier through the interaction. It was shown in [Tha13] that the prover for the sumcheck protocol can be implemented in $O(2^\mu)$ time, if the polynomial $p(\boldsymbol{X})$ can be expressed as the sum of products of multilinear polynomials.

4.4 Relation prerequisites

In this section, we define several notions of relations that are required in the rest of the paper.

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 $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$.

Pair relations. Some proof systems exhibit a gap between completeness and soundness. We formalize this using pair relations of the form $\mathcal{R} = (\mathcal{R}, \tilde{\mathcal{R}})$, where completeness holds for a relation $\mathcal{R}$, but soundness only guarantees membership in a *relaxed* relation $\tilde{\mathcal{R}} \supseteq \mathcal{R}$.

Indexed oracle pair relations. In our IORs, we consider *indexed oracle pair relations* where the instance is split into a *short instance* $\mathtt{x}$ and a list of *instance strings* $\vec{\mathtt{y}}$ written over some alphabet Σ. For notational convenience, we write $(\mathtt{i}, \mathtt{x}, \vec{\mathtt{y}}, \mathtt{w}) \in \mathcal{R}$ to denote membership in an indexed oracle relation. In the relaxed relation $\tilde{\mathcal{R}}$, we allow instance strings to be written over an augmented alphabet $\Sigma \sqcup \{\perp\}$; this will be useful when compiling IORs into non-interactive reductions.

4.5 Interactive oracle reductions

In this section we recall the notion of *interactive oracle reductions* (IORs) [BCGGRS19; BMNW25]. Intuitively, an IOR from relation $\mathcal{R}$ to relation $\mathcal{R}'$ is an interactive protocol between a prover and a verifier that reduces the verifier's task of deciding whether there exists a witness w s.t. $(x, w) \in \mathcal{R}$ to the task of

deciding whether there exists a witness w' s.t. $(x', w') \in \mathcal{R}'$ for some x' which is determined through the (randomized) execution of the protocol. We provide a formal definition below.

Let $\mathcal{R}$ and $\mathcal{R}'$ be indexed oracle pair relations. A (public-coin, holographic) interactive oracle reduction from $\mathcal{R}$ to $\mathcal{R}'$ is a tuple of polynomial-time algorithms $\mathsf{IOR} = (\mathbf{I}_{\mathsf{IOR}}, \mathbf{P}_{\mathsf{IOR}}, \mathbf{V}_{\mathsf{IOR}})$ with the following syntax.

- The indexer $\mathbf{I}_{\mathsf{IOR}}$ is a deterministic algorithm which receives as input an index $\mathbb{i}$ (for $\mathcal{R}$). It outputs a *short index* ι, *index oracle* $\vec{\mathbb{I}}$, and a new index $\mathbb{i}'$ (for $\mathcal{R}'$).
- The prover $\mathbf{P}_{\mathsf{IOR}}$ is an interactive algorithm which receives as input the index $\mathbb{i}$, short instance $\mathbb{x}$, instance oracles $\vec{\mathbb{y}}$, and witness $\mathbb{w}$. It engages in k rounds of interaction with the verifier $\mathbf{V}_{\mathsf{IOR}}$. In the i-th round, $\mathbf{P}_{\mathsf{IOR}}$ sends a proof oracle Π_i and a non-oracle message msg_i, and then receives a challenge r_i from $\mathbf{V}_{\mathsf{IOR}}$.
- The verifier $\mathbf{V}_{\mathsf{IOR}}$ is an interactive algorithm which receives as input the short index ι, short instance $\mathbb{x}$, oracle access to index oracle $\vec{\mathbb{I}}$, and oracle access to instance oracles $\vec{\mathbb{y}}$. It engages in k rounds of interaction with $\mathbf{P}_{\mathsf{IOR}}$. In the i-th round, $\mathbf{V}_{\mathsf{IOR}}$ receives oracle access to a proof oracle Π_i and a non-oracle message msg_i, and then sends a uniformly random challenge r_i.

At the end of the interaction, $\mathbf{V}_{\mathsf{IOR}}$ outputs a decision $b \in \{0, 1\}$ and a new instance $(\mathbb{x}', \vec{\mathbb{y}}')$; the new instance oracles are chosen as a function of the old instance oracles $\vec{\mathbb{y}}$ and the transcript of the interaction with the prover. The prover outputs $(b, \mathbb{x}', \vec{\mathbb{y}}', \mathbb{w}')$. We denote the IOR interaction by

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

Without loss of generality, the prover always moves first and the verifier 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. The verifier's output is a deterministic function of its input and the transcript, which we denote

$$(b, \mathbb{x}', \vec{\mathbb{y}}') \leftarrow \mathbf{V}_{\mathsf{IOR}}^{\vec{\mathbb{I}}, \vec{\mathbb{y}}, \vec{\Pi}}(\iota, \mathbb{x}, \vec{\mathsf{msg}}, \vec{r}).$$

Completeness. An IOR is said to be complete if for any $(\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}, \mathbb{w}) \in \mathcal{R}$,

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

Soundness. An IOR is said to have soundness error ε if for any $(\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}) \notin \mathcal{L}(\mathcal{R})$ and adversary $\widetilde{\mathbf{P}}$,

$$\Pr\left[b = 1 \wedge (\mathbb{i}', \mathbb{x}', \vec{\mathbb{y}}', \mathbb{w}') \in \mathcal{R}' \;\middle|\; \begin{array}{l} (\iota, \vec{\mathbb{I}}, \mathbb{i}') \leftarrow \mathbf{I}_{\mathsf{IOR}}(\mathbb{i}) \\ (b, \mathbb{x}', \vec{\mathbb{y}}', \mathbb{w}') \leftarrow \langle \widetilde{\mathbf{P}}, \mathbf{V}_{\mathsf{IOR}}^{\vec{\mathbb{I}}, \vec{\mathbb{y}}, \vec{\Pi}}(\iota, \mathbb{x}) \rangle \end{array} \right] \leq \varepsilon(\mathbb{i}, \mathbb{x}).$$

Knowledge Soundness. There are several notions of knowledge soundness for IORs. We use the notion of round-by-round knowledge soundness (RBRKS) from [BCFW25].

Definition 4.8. Let $\mathsf{IOR} = (\mathbf{P}, \mathbf{V})$ be an IOR from a relation $\mathcal{R}$ to $\mathcal{R}'$. A *knowledge state function from $\tilde{\mathcal{R}}$ to $\tilde{\mathcal{R}}'$* for IOR is a (possibly inefficient) function KState that, on input a statement $(\mathbb{x}, \mathbb{y})$, interaction transcript tr, and *knowledge state witness* w, outputs a bit and has the following properties.

- Empty transcript: if $\mathsf{tr} = \varnothing$ is the empty transcript, then $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathsf{w}) = 1$ if and only if $(\mathbb{x}, \mathbb{y}, \mathsf{w}) \in \tilde{\mathcal{R}}$.
- Prover moves: if tr is a transcript where the prover is about to move and $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathsf{w}) = 0$, then $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}||\pi, \mathsf{w}) = 0$ for every prover message π.
- Full transcript: if $\mathsf{tr} = (\pi_1, \alpha_1, \ldots, \pi_{\mathsf{k}}, \alpha_{\mathsf{k}})$ is a full transcript, then $\mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathsf{w}) = 1$ if and only if $\mathbf{V}^{\mathbb{y}, \pi_1, \ldots, \pi_{\mathsf{k}}}(\mathbb{x}, \alpha_1, \ldots, \alpha_{\mathsf{k}})$ outputs $(\mathbb{x}', \mathbb{y}')$ such that $(\mathbb{x}', \mathbb{y}', \mathsf{w}) \in \tilde{\mathcal{R}}'$.

Definition 4.9. $\mathsf{IOR} = (\mathbf{P}, \mathbf{V})$ from a relation $\mathcal{R}$ to $\mathcal{R}'$ has *round-by-round knowledge soundness errors* $(\varepsilon_1, \ldots, \varepsilon_{\mathsf{k}})$*, and extraction times* $(\mathsf{et}_1, \ldots, \mathsf{et}_{\mathsf{k}})$ if there exist a knowledge state function KState with relaxation $\tilde{\mathcal{R}}$ to $\tilde{\mathcal{R}}'$ for IOR and a deterministic extractor $\mathbf{E}_{\mathrm{rbr}}$ with the following property: for every statement $(\mathbb{x}, \mathbb{y})$, round index $i \in [\mathsf{k}]$, and interaction transcript $\mathsf{tr} = (\pi_1, \alpha_1, \ldots, \pi_{i-1}, \alpha_{i-1}, \pi_i)$ where the verifier is about to move, $\mathbf{E}_{\mathrm{rbr}}$ runs in time at most et_i and

$$\Pr_{\alpha_i}\left[\exists\, \mathsf{w} : \begin{array}{l} \mathsf{KState}\big(\mathbb{x}, \mathbb{y}, \mathsf{tr}, \mathbf{E}_{\mathrm{rbr}}(\mathbb{x}, \mathbb{y}, \mathsf{tr}||\alpha_i, \mathsf{w})\big) = 0 \\ \wedge\ \mathsf{KState}(\mathbb{x}, \mathbb{y}, \mathsf{tr}||\alpha_i, \mathsf{w}) = 1 \end{array}\right] \leq \varepsilon_i(\mathbb{x}, \mathbb{y}) \ .$$

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

We say that an IOR is RBRKS if it has round-by-round knowledge soundness errors that are negligible in the security parameter and polynomial total extraction time.

Efficiency measures. We consider the following efficiency measures for an IOR, which may be a function of the short index ι.

- *Prover time:* $T_{\mathbf{P}}$ is a function of the instance length $|\vec{\mathbb{y}}| + |\mathbb{x}|$ which represents the prover's time.
- *Verifier time:* $T_{\mathbf{V}}$ is a function of the instance length $|\vec{\mathbb{y}}| + |\mathbb{x}|$ which represents the verifier's time, *not including the time required to query the instance oracles*.
- *Communication:* s is the sum of the lengths of the non-oracle messages, $\sum_{i=1}^{\mathsf{k}} |\mathsf{msg}_i|$.
- *Query complexity:* the verifier reads q^{I} locations from the index string, $\mathsf{q}^{\mathsf{Y}_j}$ locations from the j-th instance string, and $\mathsf{q}^{\mathsf{P}_i}$ locations from the i-th proof string. Let q denote the total number of queries.

Remark 4.10 (fixed indices)**.** We sometimes write that an IOR is from $\mathcal{R}_1^{\mathbb{i}_1}$ to $\mathcal{R}_2^{\mathbb{i}_2}$, if we are only concerned with the case when the index for the input from $\mathcal{R}_1$ is always $\mathbb{i}_1$, which implies that the index for $\mathcal{R}_2$ is also fixed to $\mathbb{i}_2$ (which must be the output of the indexer on input $\mathbb{i}_1$).

Remark 4.11 (IOPPs)**.** It is easy to see that an IOPP for a relation $\mathcal{R}$ is precisely an IOR from $\mathcal{R}$ to the 'trivial' relation $\mathcal{R}_{\mathsf{true}} := \{(\mathbb{i} := \bot, \mathbb{x} := \mathsf{accept}, \vec{\mathbb{y}} := \bot, \mathbb{w} := \bot)\}$, consisting of only one element with an empty witness.

4.6 Important indexed oracle pair relations

ME. The *multilinear evaluation relation* (ME) is defined as follows.

Indexed oracle relation 4.12. The *multilinear evaluation relation* $\mathsf{ME} = (\mathcal{R}, \tilde{\mathcal{R}})$ is an indexed oracle pair relation with $\mathcal{R}$ and $\tilde{\mathcal{R}}$ both consisting of tuples of the following form:

- index: $(\mathcal{C}, \varepsilon)$, where $\mathcal{C}$ is a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$, and $\varepsilon \in (0, 1)$ is the proximity parameter,
- instance: $(\boldsymbol{z}, y)$, where $\boldsymbol{z} \in \mathbb{F}^{\log k}$ and $y \in \mathbb{F}$,
- instance oracle: $\boldsymbol{w}$, where $\boldsymbol{w} \in \mathbb{F}^n$,
- witness: $\boldsymbol{m}$, where $\boldsymbol{m} \in \mathbb{F}^k$.

such that $\hat{m}(\boldsymbol{z}) = y$ and the following distinct conditions are satisfied for $\mathcal{R}$ and $\tilde{\mathcal{R}}$ respectively:

$$\begin{aligned} \mathcal{R} : &\qquad \boldsymbol{w} = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}) \ , \\ \tilde{\mathcal{R}} : &\qquad \boldsymbol{w} \sim_{\varepsilon} \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}) \ . \end{aligned}$$

IP. The *inner product relation* (IP) is defined as follows.

Indexed oracle relation 4.13. The *inner product relation* $\mathsf{IP} = (\mathcal{R}, \tilde{\mathcal{R}})$ is an indexed oracle pair relation with $\mathcal{R}$ and $\tilde{\mathcal{R}}$ both consisting of tuples of the following form:

- index: $(\mathcal{C}, \varepsilon)$, where $\mathcal{C}$ is a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$, and $\varepsilon \in (0, 1)$ is the proximity parameter,
- instance: $(\boldsymbol{v}, \sigma)$, where $\boldsymbol{v} \in \mathbb{F}^k$ and $\sigma \in \mathbb{F}$,
- instance oracle: $\boldsymbol{w}$, where $\boldsymbol{w} \in \mathbb{F}^n$,
- witness: $\boldsymbol{m}$, where $\boldsymbol{m} \in \mathbb{F}^k$.

such that $\langle \boldsymbol{m}, \boldsymbol{v} \rangle = \sigma$ and the following distinct conditions are satisfied for $\mathcal{R}$ and $\tilde{\mathcal{R}}$ respectively:

$$\begin{aligned} \mathcal{R} : &\qquad \boldsymbol{w} = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}) \ , \\ \tilde{\mathcal{R}} : &\qquad \boldsymbol{w} \sim_{\varepsilon} \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}) \ . \end{aligned}$$

TIP. The *triple inner product relation* (TIP) is defined as follows.

Indexed oracle relation 4.14. The *triple inner product relation* $\mathsf{TIP} = (\mathcal{R}, \tilde{\mathcal{R}})$ is an indexed oracle pair relation with $\mathcal{R}$ and $\tilde{\mathcal{R}}$ both consisting of tuples of the following form:

- index: $(\mathcal{C}, \varepsilon)$, where $\mathcal{C}$ is a linear code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{C}})$, and $\varepsilon \in (0, 1)$ is the proximity parameter,
- instance: $(\boldsymbol{v}, \sigma)$, where $\boldsymbol{v} \in \mathbb{F}^k$ and $\sigma \in \mathbb{F}$,
- instance oracles: $(\boldsymbol{w}_1, \boldsymbol{w}_2)$, where $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{F}^n$,
- witness: $(\boldsymbol{m}_1, \boldsymbol{m}_2)$, where $\boldsymbol{m}_1, \boldsymbol{m}_2 \in \mathbb{F}^k$.

such that $\langle \boldsymbol{m}_1, \boldsymbol{m}_2, \boldsymbol{v} \rangle = \sigma$ and the following distinct conditions are satisfied for $\mathcal{R}$ and $\tilde{\mathcal{R}}$ respectively:

$$\begin{aligned} \mathcal{R} : &\qquad \boldsymbol{w}_1 = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}_1) \quad \wedge \quad \boldsymbol{w}_2 = \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}_2) \ , \\ \tilde{\mathcal{R}} : &\qquad \boldsymbol{w}_1 \sim_{\varepsilon} \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}_1) \quad \wedge \quad \boldsymbol{w}_2 \sim_{\varepsilon} \mathsf{Enc}_{\mathcal{C}}(\boldsymbol{m}_2) \ . \end{aligned}$$

5 A query-efficient IOPP for interleaved ERA codes

Before we describe our IOPP construction, we introduce some notation and discuss a subtlety regarding the ME relation defined over an interleaved code.

Remark 5.1 (Matrix notation)**.** In this section, we will consider alphabets of the form $(\mathbb{F}^m)^n$. We identify the elements of such an alphabet with matrices $\boldsymbol{V} \in \mathbb{F}^{m \times n}$, where each column of $\boldsymbol{V}$ is thought of as an element of $\mathbb{F}^m$. Given a matrix $\boldsymbol{V} \in \mathbb{F}^{m \times n}$ written in uppercase, we use the corresponding lowercase $\boldsymbol{v} \in \mathbb{F}^{mn}$ to denote the vector obtained by concatenating the rows of $\boldsymbol{V}$. Similarly, when m and n are clear from context, given a vector $\boldsymbol{v} \in \mathbb{F}^{mn}$, we use $\boldsymbol{V} \in \mathbb{F}^{m \times n}$ to denote the matrix such that $\boldsymbol{v}$ is the vector obtained by concatenating the rows of $\boldsymbol{V}$.

Consider the relation $\mathsf{ME}^{(\mathcal{B}^t, \varepsilon)}$, where $\mathcal{B}^t$ is an interleaved code as defined in Definition 4.2. Strictly speaking, this relation consists of tuples of the form $(\mathbb{x} = (\boldsymbol{Z}, \boldsymbol{y}), \vec{\mathbb{y}} = \boldsymbol{W}, \mathbb{w} = \boldsymbol{M})$, where $\boldsymbol{Z} \in \mathbb{F}^{t \times \log k}$, $\boldsymbol{y} \in \mathbb{F}^{t \times 1}$, $\boldsymbol{W} \in \mathbb{F}^{t \times n}$, and $\boldsymbol{M} \in \mathbb{F}^{t \times k}$. However, to make use of the distance of interleaved codes over $\mathbb{F}^t$, we slightly modify the definition of $\mathsf{ME}^{(\mathcal{B}^t, \varepsilon)}$ as follows.

Indexed oracle relation 5.2 (modified ME relation for interleaved codes)**.** A *multilinear evaluation relation* $\mathsf{ME} = (\mathcal{R}, \tilde{\mathcal{R}})$ for interleaved codes is an indexed oracle pair relation with $\mathcal{R}$ and $\tilde{\mathcal{R}}$ both consisting of tuples of the following form:

- index: $(\mathcal{B}^t, \varepsilon)$, where $\mathcal{B}$ is a code with parameters $(\mathbb{F}, n, k, \delta_{\mathcal{B}})$, and $\varepsilon \in (0, \delta_{\mathcal{B}})$ is the proximity parameter.
- instance: $(\boldsymbol{z}, y)$, where $\boldsymbol{z} \in \mathbb{F}^{\log t + \log k}$ and $y \in \mathbb{F}$,
- instance oracles: $\boldsymbol{w}$, where $\boldsymbol{w} \in \mathbb{F}^{t \cdot n}$,
- witness: $\boldsymbol{m}$, where $\boldsymbol{m} \in \mathbb{F}^{t \cdot k}$.

such that $\hat{m}(\boldsymbol{z}) = y$ and the following distinct conditions are satisfied for $\mathcal{R}$ and $\tilde{\mathcal{R}}$ respectively:

$$\mathcal{R}: \quad \boldsymbol{W} = \mathsf{Enc}_{\mathcal{B}^t}(\boldsymbol{M}) \ ,$$
$$\tilde{\mathcal{R}}: \quad \boldsymbol{W} \sim_{\varepsilon} \mathsf{Enc}_{\mathcal{B}^t}(\boldsymbol{M}) \ ,$$

where the distance is measured over the alphabet $\mathbb{F}^t$.

Remark 5.3 (Virtual oracles)**.** The IORs discussed in this paper output 'virtual oracles' that are not explicitly sent by the prover, but are linear combinations of the oracles sent by the prover. That is, the output instance oracle $\vec{\mathbb{y}}'$ is defined to be $[\![\boldsymbol{w}']\!] := \sum_{i \in [t_0]} c_i \cdot [\![\boldsymbol{w}_i]\!]$ for some coefficients $[c_i]_{i \in [t_0]}$ and oracles $([\![\boldsymbol{w}_i]\!])_{i \in [t_0]}$ previously sent by the prover. This is a common technique used implicitly or explicitly in prior work, e.g. [ACFY25; BCFW25]. Given a virtual oracle $[\![\boldsymbol{w}']\!]$ defined as above, we refer to t_0 as the *virtual blowup* of $[\![\boldsymbol{w}']\!]$, as one query to $[\![\boldsymbol{w}']\!]$ counts as t_0 many oracle queries.

Intuitively, a query to the virtual oracle $[\![\boldsymbol{w}']\!]$ is realized by the verifier querying the oracles $([\![\boldsymbol{w}_i]\!])_{i \in [t_0]}$ at the same index and then taking the appropriate linear combination of the responses. To formally capture this in our framework, one can think of the instance oracles of the output relation as being $([\![\boldsymbol{w}_i]\!])_{i \in [t_0]}$, with any claim over $\boldsymbol{w}'$ in the relation being replaced with a claim over $\sum_{i \in [t_0]} c_i \cdot \boldsymbol{w}_i$.

5.1 IOPP construction

Let $\ell \in \mathbb{N}$. For each $i \in [0, \ell - 1]$, let ERA_i be an ERA code with parameters $(\mathbb{F}, n_i, k_i, \delta_{\mathsf{ERA}_i})$ such that $k_0 > k_1 > \cdots > k_{\ell-1}$ and the base code of ERA_i is a tensor code of the form $\mathcal{B}_i = \mathcal{B}'_i \otimes \mathcal{B}'_i$, for some linear code $\mathcal{B}'_i$. Let $\mathcal{C}_{\mathsf{fin}}$ be a linear code with parameters $(\mathbb{F}, n_{\mathsf{fin}}, k_{\mathsf{fin}}, \delta_{\mathcal{C}_{\mathsf{fin}}})$ such that $k_{\mathsf{fin}} < k_{\ell-1}$ and there exists

an IOPP, $\mathsf{IOPP}_{\mathsf{fin}}$, for $\mathsf{ME}^{(\mathcal{C}_{\mathsf{fin}},\varepsilon_{\mathsf{fin}})}$. Finally, for each $i \in [0, \ell-1]$, let $\varepsilon_i \in (0, \delta_{\mathsf{ERA}_i})$ and $\varepsilon_{\mathsf{fin}} \in (0, \delta_{\mathcal{C}_{\mathsf{fin}}})$ be proximity parameters.

We describe an IOPP for $\mathsf{ME}^{(\mathsf{ERA}_0^{t_0},\varepsilon_0)}$. Since $\mathsf{ME}^{(\mathsf{ERA}_0^{t_0},\varepsilon_0)}$ considers messages of length $t_0 \cdot k_0$, when defined as a vector over $\mathbb{F}$, one should think of the message size of this relation as $k := t_0 \cdot k_0$ rather than k_0. The IOPP is obtained by composing $\mathsf{StartingIOR}$, the reductions $\mathsf{Reduce}_1, \ldots, \mathsf{Reduce}_\ell$, and finally $\mathsf{IOPP}_{\mathsf{fin}}$. We formalize this composition in Fig. 2, and describe $\mathsf{StartingIOR}$ and Reduce_i in Section 5.2 and Section 6 respectively. We summarize the resulting efficiency in the following theorem.

Indexer input: $(\mathsf{ERA}_0^{t_0}, \varepsilon_0)$.
Indexer output: For each $i \in [\ell]$, the indexer for Reduce_i is run on the index $(\mathsf{ERA}_{i-1}, \varepsilon_{i-1})$. Then, the indexer for $\mathsf{IOPP}_{\mathsf{fin}}$ is run on $(\mathcal{C}_{\mathsf{fin}}, \varepsilon_{\mathsf{fin}})$. The top-level indexer outputs the collection of indexer data required by these component protocols.

Input instance: $\mathbb{x} := (\boldsymbol{z}, y)$ where $\boldsymbol{z} \in \mathbb{F}^{\log t_0 + \log k_0}$ and $y \in \mathbb{F}$.
Input instance oracles: $\vec{\mathbb{y}} := \boldsymbol{w} \in \mathbb{F}^{t_0 \cdot n_0}$.
Input witness: $\boldsymbol{m} \in \mathbb{F}^{t_0 \cdot k_0}$.

$\langle \mathbf{P}(\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}, \mathbb{w}), \mathbf{V}^{\vec{\mathbb{I}},\vec{\mathbb{y}}}(\iota, \mathbb{x}) \rangle$:

1. $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{StartingIOR}$ on the input claim, and output an $\mathsf{ME}^{(\mathsf{ERA}_0,\varepsilon_0)}$ claim.
2. For each $i \in [\ell - 1]$, $\mathbf{P}$ and $\mathbf{V}$ run Reduce_i on the $\mathsf{ME}^{(\mathsf{ERA}_{i-1},\varepsilon_{i-1})}$ claim, and output an $\mathsf{ME}^{(\mathsf{ERA}_i,\varepsilon_i)}$ claim.
3. $\mathbf{P}$ and $\mathbf{V}$ run Reduce_ℓ on the $\mathsf{ME}^{(\mathsf{ERA}_{\ell-1},\varepsilon_{\ell-1})}$ claim, and output an $\mathsf{ME}^{(\mathcal{C}_{\mathsf{fin}},\varepsilon_{\mathsf{fin}})}$ claim.
4. $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{IOPP}_{\mathsf{fin}}$ on the final $\mathsf{ME}^{(\mathcal{C}_{\mathsf{fin}},\varepsilon_{\mathsf{fin}})}$ claim.
5. $\mathbf{V}$ outputs whatever $\mathsf{IOPP}_{\mathsf{fin}}$ outputs.

Figure 2: Top-level IOPP for $\mathsf{ME}^{(\mathsf{ERA}_0^{t_0},\varepsilon_0)}$.

Theorem 5.4. *If* $\mathsf{IOPP}_{\mathsf{fin}}$ *is an RBRKS IOPP for* $\mathsf{ME}^{(\mathcal{C}_{\mathsf{fin}},\varepsilon_{\mathsf{fin}})}$*, then the protocol in Fig. 2 is an RBRKS IOPP for* $\mathsf{ME}^{(\mathsf{ERA}_0^{t_0},\varepsilon_0)}$ *whose efficiency parameters are the sum of those of its component IORs* $\mathsf{StartingIOR}$, $\mathsf{Reduce}_1, \ldots, \mathsf{Reduce}_\ell$, *and* $\mathsf{IOPP}_{\mathsf{fin}}$.

Proof. It is shown in Lemma 5.5 and Lemma 6.1 that $\mathsf{StartingIOR}$ and each Reduce_i are RBRKS. The theorem follows from the fact that RBRKS IORs are (defined to be) well-behaved under composition [BMNW25; BCFW25]. □

5.2 Construction of $\mathsf{StartingIOR}$

We present the construction of $\mathsf{StartingIOR}$ from $\mathsf{ME}^{(\mathsf{ERA}^{t_0},\varepsilon_0)}$ to $\mathsf{ME}^{(\mathsf{ERA},\varepsilon_0)}$ in Fig. 3, and summarize its properties in the following lemma.

Lemma 5.5. *Let* ERA *be an* ERA *code with parameters* $(\mathbb{F}, n, k, \delta_{\mathsf{ERA}})$*, and let* $t_0 \in \mathbb{N}$ *be an interleaving parameter.* $\mathsf{StartingIOR}$ *is an RBRKS IOR from* $\mathsf{ME}^{(\mathsf{ERA}^{t_0},\varepsilon_0)}$ *to* $\mathsf{ME}^{(\mathsf{ERA},\varepsilon_0)}$ *with the following properties:*

prover time	*verifier time*	*queries*	*comm. ($\mathbb{F}$-elements)*	*virtual blowup of output*
$O(t_0 n)$	$O(\log t_0 + \log n)$	0	$3 \log t_0 + 3 \log k$	t_0

Proof. The efficiency properties of StartingIOR are immediate from Fig. 3. Furthermore, the verifier does not explicitly query any oracles; instead, it outputs 'virtual oracles' that are linear combinations of the oracles sent by the prover (see Remark 5.3).

The RBRKS proof follows similarly to [BCFW25, Theorem 6.2] (while skipping the initial zero-check). This shows that the protocol has round-by-round knowledge soundness, with knowledge soundness error:

$$\varepsilon_{\mathsf{mca}}(\mathsf{ERA}^{t_0}, t_0, \varepsilon_0) + \frac{2 \cdot |\Lambda\left(\mathsf{ERA}^{t_0}, \varepsilon_0\right)|}{|\mathbb{F}|} \quad ,$$

where $\varepsilon_{\mathsf{mca}}(\mathsf{ERA}^{t_0}, t_0, \varepsilon_0)$ is the mutual correlated agreement error defined in Definition 4.5. □

Indexer input: $(\mathsf{ERA}^{t_0}, \varepsilon_0)$.
Indexer output: $(\mathbb{i}', \iota, \vec{\mathbb{I}}) := ((\mathsf{ERA}, \varepsilon_0), \bot, \bot)$.

Input instance: $\mathbb{x} := (\boldsymbol{z}, y)$ where $\boldsymbol{z} \in \mathbb{F}^{\log t_0 + \log k}$, and $y \in \mathbb{F}$.
Input instance oracles: $\vec{\mathbb{y}} := \boldsymbol{w} \in \mathbb{F}^{t_0 \cdot n}$.
Input witness: $\boldsymbol{m} \in \mathbb{F}^{t_0 \cdot k}$.

$\langle \mathbf{P}(\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}, \mathbb{w}), \mathbf{V}^{\vec{\mathbb{I}}, \vec{\mathbb{y}}}(\iota, \mathbb{x}) \rangle$:

1. $\mathbf{P}$ and $\mathbf{V}$ parse the oracle $[\![\boldsymbol{w}]\!]$ as consisting of t_0 many oracles $[\![\boldsymbol{W}[1, \cdot]]\!], \ldots, [\![\boldsymbol{W}[t_0, \cdot]]\!]$, consisting of the rows of the matrix $\boldsymbol{W} \in \mathbb{F}^{t_0 \times n}$.
2. $\mathbf{P}$ parses $\boldsymbol{m} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{t_0})$, where $\boldsymbol{m}_i \in \mathbb{F}^k$ for each $i \in [t_0]$.
3. $\mathbf{P}$ and $\mathbf{V}$ 'randomize' the evaluation point $\boldsymbol{z}$ by engaging in the sumcheck protocol to reduce checking that $\sum_{\boldsymbol{b} \in \mathbb{F}^{\log t_0 + \log k}} \mathrm{eq}(\boldsymbol{z}, \boldsymbol{b}) \cdot \hat{m}(\boldsymbol{b}) = y$ to checking that $\mathrm{eq}(\boldsymbol{z}, \boldsymbol{r}) \cdot \hat{m}(\boldsymbol{r}) = y_{\boldsymbol{r}}$.
4. $\mathbf{P}$ and $\mathbf{V}$ parse $\boldsymbol{r} = (\boldsymbol{r}_1, \boldsymbol{r}_2)$, where $\boldsymbol{r}_1 \in \mathbb{F}^{\log t_0}$ and $\boldsymbol{r}_2 \in \mathbb{F}^{\log k}$.
5. Define

$$\begin{aligned}
\mathbb{x}' &:= (\boldsymbol{r}_2, y' := y_{\boldsymbol{r}} \cdot \mathrm{eq}(\boldsymbol{z}, \boldsymbol{r})^{-1}), \\
\vec{\mathbb{y}}' &:= \sum_{i=1}^{t_0} \mathrm{eq}(\boldsymbol{r}_1, \langle i - 1 \rangle) \cdot [\![\boldsymbol{W}[i, \cdot]]\!], \\
\mathbb{w}' &:= \sum_{i=1}^{t_0} \mathrm{eq}(\boldsymbol{r}_1, \langle i - 1 \rangle) \cdot \boldsymbol{m}_i \quad .
\end{aligned}$$

6. $\mathbf{P}$ outputs $(\mathbb{x}', \vec{\mathbb{y}}', \mathbb{w}')$, and $\mathbf{V}$ outputs $(\mathbb{x}', \vec{\mathbb{y}}')$.

Figure 3: The StartingIOR IOR from $\mathsf{ME}^{(\mathsf{ERA}^{t_0}, \varepsilon_0)}$ to $\mathsf{ME}^{(\mathsf{ERA}, \varepsilon_0)}$.

6 The Reduce IOR

In this section, we use codeswitching techniques from [BMMS25; BCFRRZ25] to construct our main Reduce IOR from $\mathsf{ME}^{(\mathsf{ERA},\varepsilon)}$ to $\mathsf{ME}^{(\mathcal{C}',\varepsilon')}$, where:

- $\mathsf{ERA} := \mathsf{ERA}(\mathbb{F}, \mathcal{B}, r, \boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{M})$ has parameters $(\mathbb{F}, n_{\mathsf{ERA}}, k, \delta_{\mathsf{ERA}})$. Define $\pi_1, \pi_2 : [n_{\mathsf{ERA}}] \to [n_{\mathsf{ERA}}]$ as the permutations corresponding to the permutation matrices $\boldsymbol{P}_1, \boldsymbol{P}_2$ respectively, and define $\boldsymbol{v} \in \mathbb{F}^{n_{\mathsf{ERA}}}$ to be the vector corresponding to the diagonal entries of $\boldsymbol{M}$.
- $\mathcal{B}$ is a base code with parameters $(\mathbb{F}, n_{\mathcal{B}}, k, \delta_{\mathcal{B}})$, such that $\mathcal{B} = \mathcal{B}' \otimes \mathcal{B}'$ for some linear code $\mathcal{B}'$ with parameters $(\mathbb{F}, \sqrt{n_{\mathcal{B}}}, \sqrt{k}, \sqrt{\delta_{\mathcal{B}}})$.
- $\mathcal{C}'$ is an output code with parameters $(\mathbb{F}, n_{\mathcal{C}'}, k', \delta_{\mathcal{C}'})$.

Throughout this section, we use the following four quantities:

$$\ell_{\boldsymbol{m}} := \lceil k/k' \rceil, \ell_{\mathcal{B}} := \lceil n_{\mathcal{B}}/k' \rceil, \ell_{\boldsymbol{g}} := \left\lceil \sqrt{n_{\mathcal{B}} k}/k' \right\rceil, \ell_{\mathsf{ERA}} := \lceil n_{\mathsf{ERA}}/k' \rceil \ .$$

Figure 4 presents the construction of Reduce.

Indexer input: $(\mathsf{ERA}, \varepsilon)$.
Indexer output: The indexer runs the helper function $\mathsf{CodeswitchOracles}(\mathsf{ERA})$ (see Fig. 5) to generate the index oracles $(\boldsymbol{w}_1^{\mathsf{ind}}, \ldots, \boldsymbol{w}_{\ell_{\mathsf{ind}}}^{\mathsf{ind}})$ required in the codeswitch steps, where $\boldsymbol{w}_i^{\mathsf{ind}} \in \mathbb{F}^{n_{\mathcal{C}'}}$ for each $i \in [\ell_{\mathsf{ind}}]$. The indexer outputs $(\mathbb{i}', \iota, \vec{\mathbb{I}}) := ((\mathcal{C}', \varepsilon'), \bot, (\boldsymbol{w}_1^{\mathsf{ind}}, \ldots, \boldsymbol{w}_{\ell_{\mathsf{ind}}}^{\mathsf{ind}}))$.

Input instance: $\mathbb{x} := (\boldsymbol{z}, y)$ where $\boldsymbol{z} \in \mathbb{F}^{\log k}$, and $y \in \mathbb{F}$.
Input instance oracles: $\vec{\mathbb{y}} := \boldsymbol{w} \in \mathbb{F}^{n_{\mathsf{ERA}}}$, where $\mathbf{V}$ has virtual oracle access to $\boldsymbol{w}$ and the virtual blowup of $[\![\boldsymbol{w}]\!]$ is t_0 (see Remark 5.3).
Input witness: $\boldsymbol{m} \in \mathbb{F}^k$.

$\langle \mathbf{P}(\mathbb{i}, \mathbb{x}, \vec{\mathbb{y}}, \mathbb{w}), \mathbf{V}^{\vec{\mathbb{I}}, \vec{\mathbb{y}}}(\iota, \mathbb{x}) \rangle$:

1. $\mathbf{P}$ parses $\boldsymbol{m} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{\ell_{\boldsymbol{m}}})$, such that $\boldsymbol{m}_i \in \mathbb{F}^{k'}$, for each $i \in [\ell_{\boldsymbol{m}}]$.
2. Let $\boldsymbol{z} = (\boldsymbol{z}_1, \boldsymbol{z}_2)$, where $\boldsymbol{z}_1 \in \mathbb{F}^{\log k - \log k'}$ and $\boldsymbol{z}_2 \in \mathbb{F}^{\log k'}$.
3. For each $i \in [\ell_{\boldsymbol{m}}]$, $\mathbf{P}$ computes $\boldsymbol{w}_i := \mathsf{Enc}_{\mathsf{ERA}}(\boldsymbol{m}_i)$ and $y_i^{\mathsf{eval}} := \hat{m}_i(\boldsymbol{z})$.
4. $\mathbf{P}$ sends $([\![\boldsymbol{w}_i]\!])_{i \in [\ell_{\boldsymbol{m}}]}$ and $[y_i^{\mathsf{eval}}]_{i \in [\ell_{\boldsymbol{m}}]}$ to $\mathbf{V}$.
5. $\mathbf{V}$ checks that $\sum_{\boldsymbol{b} \in \{0,1\}^{\log k - \log k'}} \mathrm{eq}(\boldsymbol{z}_1, \boldsymbol{b}) \cdot y_i^{\mathsf{eval}} = y$.
6. $\mathbf{V}$ samples $\boldsymbol{z}_i^{\mathsf{ood}} \xleftarrow{\$} \mathbb{F}^{\log k'}$ for each $i \in [\ell_{\boldsymbol{m}}]$ and sends $[\boldsymbol{z}_i^{\mathsf{ood}}]_{i \in [\ell_{\boldsymbol{m}}]}$ to $\mathbf{P}$.
7. $\mathbf{P}$ sends $[y_i^{\mathsf{ood}} := \hat{m}_i(\boldsymbol{z}_i^{\mathsf{ood}})]_{i \in [\ell_{\boldsymbol{m}}]}$ to $\mathbf{V}$.

Generating codeswitch claims

8. $\mathbf{V}$ samples $\nu := \left\lceil \frac{-\lambda}{\log(1-\varepsilon)} \right\rceil$ indices $\alpha_1, \ldots, \alpha_\nu \xleftarrow{\$} \mathbb{F}^{\log n}$ and sends them to $\mathbf{P}$.
9. For each $j \in [\nu]$, $\mathbf{V}$ queries $[\![\boldsymbol{w}]\!]$ at index α_j to obtain $\sigma_j^{\mathsf{cs}} := \boldsymbol{w}[\alpha_j]$.
10. $\mathbf{P}$ and $\mathbf{V}$ engage in the subprotocol $\mathsf{CodeswitchClaims}$, the details of which are presented in Sections 6.2 and 6.3. At a high-level, $\mathsf{CodeswitchClaims}$ allows $\mathbf{P}$ to convince $\mathbf{V}$ that, for the $\boldsymbol{m}$ underlying $[\boldsymbol{w}_i]_{i \in [\ell_{\boldsymbol{m}}]}$, the α_j-th index of $\mathsf{Enc}_{\mathsf{ERA}}(\boldsymbol{m})$ is σ_j^{cs}, for all $j \in [\nu]$. The interaction of $\mathbf{P}$

and $\mathbf{V}$ while running CodeswitchClaims can be summarized as follows:

- Both $\mathbf{P}$ and $\mathbf{V}$ know certain integer parameters $\mathsf{numIP}, \mathsf{numTIP}, \ell_{\mathsf{aux}}$ (described in Fig. 6).
- Over the course of the subprotocol, $\mathbf{P}$ sends oracles $([\![\boldsymbol{w}_i^{\mathsf{aux}}]\!])_{i\in[\ell_{\mathsf{aux}}]}$ to $\mathbf{V}$.
- For each $i \in [\mathsf{numIP}]$, $\mathbf{P}$ and $\mathbf{V}$ agree on an oracle $[\![\boldsymbol{w}_i^{\mathsf{IP}}]\!]$ chosen from the set $\big(([\![\boldsymbol{w}_i^{\mathsf{ind}}]\!])_{i\in[\ell_{\mathsf{ind}}]}, ([\![\boldsymbol{w}_i]\!])_{i\in[\ell_{\boldsymbol{m}}]}, ([\![\boldsymbol{w}_i^{\mathsf{aux}}]\!])_{i\in[\ell_{\mathsf{aux}}]}\big)$, as well as a vector $\boldsymbol{v}_i^{\mathsf{IP}} \in \mathbb{F}^{k'}$ and a scalar $\sigma_i^{\mathsf{IP}} \in \mathbb{F}$, with the claim that

$$((\boldsymbol{v}_i^{\mathsf{IP}}, \sigma_i^{\mathsf{IP}}), \boldsymbol{w}_i^{\mathsf{IP}}, \boldsymbol{m}_i^{\mathsf{IP}}) \in \mathsf{IP}^{(\mathcal{C}',\varepsilon')} \ ,$$

  where $\boldsymbol{m}_i^{\mathsf{IP}} \in \mathbb{F}^{k'}$ is a witness known to $\mathbf{P}$.
- For each $i \in [\mathsf{numTIP}]$, $\mathbf{P}$ and $\mathbf{V}$ agree on two oracles $[\![\boldsymbol{w}_{i,1}^{\mathsf{TIP}}]\!], [\![\boldsymbol{w}_{i,2}^{\mathsf{TIP}}]\!]$ chosen from the set $\big(([\![\boldsymbol{w}_i^{\mathsf{ind}}]\!])_{i\in[\ell_{\mathsf{ind}}]}, ([\![\boldsymbol{w}_i]\!])_{i\in[\ell_{\boldsymbol{m}}]}, ([\![\boldsymbol{w}_i^{\mathsf{aux}}]\!])_{i\in[\ell_{\mathsf{aux}}]}\big)$, as well as a vector $\boldsymbol{v}_i^{\mathsf{TIP}} \in \mathbb{F}^{k'}$ and a scalar $\sigma_i^{\mathsf{TIP}} \in \mathbb{F}$, with the claim that

$$((\boldsymbol{v}_i^{\mathsf{TIP}}, \sigma_i^{\mathsf{TIP}}), (\boldsymbol{w}_{i,1}^{\mathsf{TIP}}, \boldsymbol{w}_{i,2}^{\mathsf{TIP}}), (\boldsymbol{m}_{i,1}^{\mathsf{TIP}}, \boldsymbol{m}_{i,2}^{\mathsf{TIP}})) \in \mathsf{TIP}^{(\mathcal{C}',\varepsilon')} \ ,$$

  where $\boldsymbol{m}_{i,1}^{\mathsf{TIP}}, \boldsymbol{m}_{i,2}^{\mathsf{TIP}} \in \mathbb{F}^{k'}$ are witnesses known to $\mathbf{P}$.

Running sumcheck on the claims produced by CodeswitchClaims

11. $\mathbf{V}$ samples the random challenge $\beta \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathbf{P}$.
12. Define

$$f(\boldsymbol{X}) := \sum_{i=1}^{\ell_{\boldsymbol{m}}} \beta^i \cdot \hat{m}_i(\boldsymbol{X}) \cdot \mathrm{eq}(\boldsymbol{z}_2, \boldsymbol{X}) + \sum_{i=1}^{\ell_{\boldsymbol{m}}} \beta^{\ell_{\boldsymbol{m}}+i} \cdot \hat{m}_i(\boldsymbol{X}) \cdot \mathrm{eq}(\boldsymbol{z}_i^{\mathsf{ood}}, \boldsymbol{X})$$
$$+ \sum_{i=1}^{\mathsf{numIP}} \beta^{2\ell_{\boldsymbol{m}}+i} \cdot \hat{m}_i^{\mathsf{IP}}(\boldsymbol{X}) \cdot \hat{v}_i^{\mathsf{IP}}(\boldsymbol{X}) + \sum_{i=1}^{\mathsf{numTIP}} \beta^{2\ell_{\boldsymbol{m}}+\mathsf{numIP}+i} \cdot \hat{m}_{i,1}^{\mathsf{TIP}}(\boldsymbol{X}) \cdot \hat{m}_{i,2}^{\mathsf{TIP}}(\boldsymbol{X}) \cdot \hat{v}_i^{\mathsf{TIP}}(\boldsymbol{X})$$

13. Also define

$$\sigma := \sum_{i=1}^{\ell_{\boldsymbol{m}}} \beta^i \cdot y_i^{\mathsf{eval}} + \sum_{i=1}^{\ell_{\boldsymbol{m}}} \beta^{\ell_{\boldsymbol{m}}+i} \cdot y_i^{\mathsf{ood}} + \sum_{i=1}^{\mathsf{numIP}} \beta^{2\ell_{\boldsymbol{m}}+i} \cdot \sigma_i^{\mathsf{IP}} + \sum_{i=1}^{\mathsf{numTIP}} \beta^{2\ell_{\boldsymbol{m}}+\mathsf{numIP}+i} \cdot \sigma_i^{\mathsf{TIP}} \ .$$

14. $\mathbf{P}$ and $\mathbf{V}$ engage in the sumcheck protocol to prove that $\sum_{\boldsymbol{b}\in\mathbb{F}^{\log k'}} f(\boldsymbol{b}) = \sigma$, yielding the value $y_{\boldsymbol{r}} \in \mathbb{F}$ and $\mathbf{V}$'s challenges $\boldsymbol{r} \in \mathbb{F}^{\log k'}$, such that $\mathbf{P}$ is just left to convince $\mathbf{V}$ that $f(\boldsymbol{r}) = y_{\boldsymbol{r}}$.

Proving the sumcheck evaluation claim via individual evaluation claims

15. $\mathbf{P}$ sends the following purported values to $\mathbf{V}$:

$$[a_i^{\mathsf{eval}}]_{i=1}^{\ell_{\boldsymbol{m}}} := [\hat{m}_i(\boldsymbol{r})]_{i=1}^{\ell_{\boldsymbol{m}}}, \qquad [a_i^{\mathsf{IP}}]_{i=1}^{\mathsf{numIP}} := [\hat{m}_i^{\mathsf{IP}}(\boldsymbol{r})]_{i=1}^{\mathsf{numIP}},$$
$$[a_{i,1}^{\mathsf{TIP}}]_{i=1}^{\mathsf{numTIP}} := [\hat{m}_{i,1}^{\mathsf{TIP}}(\boldsymbol{r})]_{i=1}^{\mathsf{numTIP}}, \quad \text{and} \quad [a_{i,2}^{\mathsf{TIP}}]_{i=1}^{\mathsf{numTIP}} := [\hat{m}_{i,2}^{\mathsf{TIP}}(\boldsymbol{r})]_{i=1}^{\mathsf{numTIP}} \ .$$

16. $\mathbf{V}$ defines $[a_i^{\mathsf{ood}}]_{i=1}^{\ell_{\boldsymbol{m}}} := [a_i^{\mathsf{eval}}]_{i=1}^{\ell_{\boldsymbol{m}}}$, and checks that

$$y_{\boldsymbol{r}} = \sum_{i=1}^{\ell_{\boldsymbol{m}}} \beta^i \cdot a_i^{\mathsf{eval}} \cdot \mathrm{eq}(\boldsymbol{z}_2, \boldsymbol{r}) + \sum_{i=1}^{\ell_{\boldsymbol{m}}} \beta^{\ell_{\boldsymbol{m}}+i} \cdot a_i^{\mathsf{ood}} \cdot \mathrm{eq}(\boldsymbol{z}_i^{\mathsf{ood}}, \boldsymbol{r}) + \sum_{i=1}^{\mathsf{numIP}} \beta^{2\ell_{\boldsymbol{m}}+i} \cdot a_i^{\mathsf{IP}} \cdot \hat{v}_i^{\mathsf{IP}}(\boldsymbol{r}) + \sum_{i=1}^{\mathsf{numTIP}} \beta^{2\ell_{\boldsymbol{m}}+\mathsf{numIP}+i} \cdot a_{i,1}^{\mathsf{TIP}} \cdot a_{i,2}^{\mathsf{TIP}} \cdot \hat{v}_i^{\mathsf{IP}}(\boldsymbol{r}) \ .$$

Batching the individual evaluation claims

17. $\mathbf{V}$ samples the random challenge $\gamma \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathbf{P}$.
18. Define

$$y' := \sum_{i=1}^{\ell_{\boldsymbol{m}}} \gamma^i \cdot a_i^{\mathsf{eval}} + \sum_{i=1}^{\ell_{\boldsymbol{m}}} \gamma^{\ell_{\boldsymbol{m}}+i} \cdot a_i^{\mathsf{ood}} + \sum_{i=1}^{\mathsf{numIP}} \gamma^{2\ell_{\boldsymbol{m}}+i} \cdot a_i^{\mathsf{IP}} + \sum_{i=1}^{\mathsf{numTIP}} \gamma^{2\ell_{\boldsymbol{m}}+\mathsf{numIP}+i} \cdot a_{i,1}^{\mathsf{TIP}} + \sum_{i=1}^{\mathsf{numTIP}} \gamma^{2\ell_{\boldsymbol{m}}+\mathsf{numIP}+i} \cdot a_{i,2}^{\mathsf{TIP}} \ .$$

19. $\mathbf{P}$ and $\mathbf{V}$ define the virtual oracle

$$[\![\boldsymbol{w}']\!] := \sum_{i=1}^{\ell_{\boldsymbol{m}}} \gamma^i \cdot [\![\boldsymbol{w}_i]\!] + \sum_{i=1}^{\ell_{\boldsymbol{m}}} \gamma^{\ell_{\boldsymbol{m}}+i} \cdot [\![\boldsymbol{w}_i]\!] + \sum_{i=1}^{\mathsf{numIP}} \gamma^{2\ell_{\boldsymbol{m}}+i} \cdot [\![\boldsymbol{w}_i^{\mathsf{IP}}]\!] + \sum_{i=1}^{\mathsf{numTIP}} \gamma^{2\ell_{\boldsymbol{m}}+\mathsf{numIP}+i} \cdot [\![\boldsymbol{w}_{i,1}^{\mathsf{TIP}}]\!] + \sum_{i=1}^{\mathsf{numTIP}} \gamma^{2\ell_{\boldsymbol{m}}+\mathsf{numIP}+i} \cdot [\![\boldsymbol{w}_{i,2}^{\mathsf{TIP}}]\!] \ ,$$

and $\mathbf{P}$ defines $\boldsymbol{m}'$ to be the underlying message obtained by computing an analogous linear combination of $\boldsymbol{m}_i, \boldsymbol{m}_i^{\mathsf{IP}}, \boldsymbol{m}_{i,1}^{\mathsf{TIP}}, \boldsymbol{m}_{i,2}^{\mathsf{TIP}}$.

20. $\mathbf{P}$ and $\mathbf{V}$ batch the individual evaluation claims (at the point $\boldsymbol{r}$) by defining

$$\mathbb{x}' := (\boldsymbol{r}, y'), \quad \vec{\mathbb{y}}' := \boldsymbol{w}', \quad \mathbb{w}' := \boldsymbol{m}' \ .$$

21. $\mathbf{P}$ outputs $(\mathbb{x}', \vec{\mathbb{y}}', \mathbb{w}')$, and $\mathbf{V}$ outputs $(\mathbb{x}', \vec{\mathbb{y}}')$.

Figure 4: The Reduce IOR from $\mathsf{ME}^{(\mathsf{ERA},\varepsilon)}$ to $\mathsf{ME}^{(\mathcal{C}',\varepsilon')}$.

Lemma 6.1. *Let* $\mathsf{numIP}, \mathsf{numTIP} \in \mathbb{N}$ *be the number of inner product claims and triple inner product claims respectively in* CodeswitchClaims. *Let* $s := 3\ell_{\boldsymbol{m}} + \mathsf{numIP} + 2 \cdot \mathsf{numTIP}$, *let* t_{in} *be the virtual blowup of* $[\![\boldsymbol{w}]\!]$, *and let* $t_{\mathsf{out}} := \ell_{\boldsymbol{m}} + \ell_{\mathcal{B}} + \ell_{\boldsymbol{g}} + 5\ell_{\mathsf{ERA}}$ *be the virtual blowup of the output oracles. If* CodeswitchClaims *has the following efficiency parameters:*

prover time	*verifier time*	*queries*	*comm. (*$\mathbb{F}$*-elements)*
$O(\ell_{\mathsf{ERA}} \cdot n_{\mathsf{ERA}} + t_{\mathsf{out}} \cdot T_{\mathcal{C}'})$	$O(\log k' + s)$	0	C_{cs}

then Reduce *is an RBRKS IOR from* $\mathsf{ME}^{(\mathsf{ERA},\varepsilon)}$ *to* $\mathsf{ME}^{(\mathcal{C}',\varepsilon')}$ *with the following properties:*

prover time	*verifier time*	*queries*	*comm. ($\mathbb{F}$-elements)*	*virtual blowup of output*
$O(\ell_{\mathsf{ERA}} \cdot n_{\mathsf{ERA}} + t_{\mathsf{out}} \cdot T_{\mathcal{C}'})$	$O\left(\log k' + s + \frac{-\lambda t_{\mathsf{in}}}{\log(1-\varepsilon)}\right)$	$\left\lceil \frac{-\lambda t_{\mathsf{in}}}{\log(1-\varepsilon)} \right\rceil$	$\mathsf{C}_{\mathsf{cs}} + 4\log k' + s$	t_{out}

Proof. The prover time, verifier time, and number of queries are immediate from Fig. 4. The communication overhead can be summarized as follows:

- $4 \log k'$ field elements are sent for the sumcheck of the polynomial defined in Item 12.
- $\ell_{\boldsymbol{m}}$ field elements are sent in Item 3 for the evaluation claims.
- $\ell_{\boldsymbol{m}}$ field elements are sent in Item 7 for the out-of-domain claims.
- $\ell_{\boldsymbol{m}} + \mathsf{numIP} + 2 \cdot \mathsf{numTIP}$ field elements are sent in Item 15.

RBRKS follows by adapting the arguments from prior work, in particular the PIOP-to-IOR compilation in [ACFY25; BFRW25] and the codeswitch analysis in [BCFW25]. We discuss the soundness of the PIOP corresponding to Reduce in Section 6.3. The virtual blowup of the output is dictated by the number of unique oracles output by CodeswitchClaims, which is $\ell_{\boldsymbol{m}} + \ell_{\mathcal{B}} + \ell_{\boldsymbol{g}} + 5\ell_{\mathsf{ERA}}$, as outlined in detail in Section 6.5. The exact value of C_{cs} is also discussed in Section 6.5. □

Remark 6.2 (Out-of-domain samples)**.** In Reduce, we impose evaluation claims of the form $\hat{m}(\boldsymbol{z}_i^{\mathsf{ood}}) = y_i^{\mathsf{ood}}$ for each $i \in [\ell_{\boldsymbol{m}}]$, where each $\boldsymbol{z}_i^{\mathsf{ood}} \xleftarrow{\$} \mathbb{F}^{\log k'}$. These are known as out-of-domain samples, which were first introduced in the context of Reed–Solomon codes in [BGKS20], and generalized to arbitrary codes in [BCFW25]. Our techniques are a slight variant of those in [BCFW25]. Informally, out-of-domain sampling enables efficient proximity testing in the list decoding regime: even if there are polynomially many codewords within distance ε of a word $\boldsymbol{w}$ sent as an oracle, with high probability over the choice of an evaluation point $\boldsymbol{z}^{\mathsf{ood}}$ their underlying messages evaluate to distinct values at $\boldsymbol{z}^{\mathsf{ood}}$.

6.1 Prerequisites for codeswitching

We present the details of CodeswitchOracles and CodeswitchClaims shortly in Section 6.3, but first we describe some useful preliminaries.

SplitAndEncode. Looking ahead, we want to reduce to a claim about $\mathcal{C}'$ while encoding every relevant vector, *regardless of its length*, using $\mathcal{C}'$. Since $\mathcal{C}'$ has message length k', this requires splitting vectors into length-k' blocks and encoding each block separately under $\mathcal{C}'$. We therefore define the following helper function SplitAndEncode for a vector $\boldsymbol{m} \in \mathbb{F}^{\mathsf{len}}$.

SplitAndEncode($\boldsymbol{m}$) :
1. Parse $\boldsymbol{m} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_\ell)$ where each $\boldsymbol{m}_i \in \mathbb{F}^{k'}$ and $\ell := \mathsf{len}/k'$.
2. For each $i \in [\ell]$, compute $\boldsymbol{w}_i := \mathsf{Enc}_{\mathcal{C}'}(\boldsymbol{m}_i)$.
3. Output $([\boldsymbol{m}_i]_{i\in[\ell]}, [\boldsymbol{w}_i]_{i\in[\ell]})$.

SplitClaimIP and SplitClaimTIP. Consider the claim $\langle \boldsymbol{m}, \boldsymbol{m}', \boldsymbol{p} \rangle = \sigma$ for public $\boldsymbol{p} \in \mathbb{F}^{\mathsf{len}}$ and $\sigma \in \mathbb{F}$, and 'split-committed' vectors $\boldsymbol{m}, \boldsymbol{m}' \in \mathbb{F}^{\mathsf{len}}$. To prove these claims, we define and describe SplitClaimTIP below. Again, since $\mathcal{C}'$ has message length k', it can natively only support inner product claims with vectors of length k', so we need to split the vectors into length k' parts, prove the inner product claim for each part separately, and check that the sum of the inner product claims for each part is equal to σ.

$\mathsf{SplitClaimTIP}(\boldsymbol{m}, \boldsymbol{m}', \boldsymbol{p}, \sigma, (\llbracket \boldsymbol{w}_i \rrbracket)_{i\in[\ell]}, (\llbracket \boldsymbol{w}'_i \rrbracket)_{i\in[\ell]})$:

1. Parse $\boldsymbol{m} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_\ell)$ and $\boldsymbol{m}' = (\boldsymbol{m}'_1, \ldots, \boldsymbol{m}'_\ell)$ where each $\boldsymbol{m}_i, \boldsymbol{m}'_i \in \mathbb{F}^{k'}$.
2. Parse $\boldsymbol{p} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\ell)$ where each $\boldsymbol{p}_i \in \mathbb{F}^{k'}$.
3. For each $i \in [\ell]$, $\mathbf{P}$ computes $\sigma_i := \langle \boldsymbol{m}_i, \boldsymbol{m}'_i, \boldsymbol{p}_i \rangle$ and sends it to $\mathbf{V}$.
4. $\mathbf{V}$ checks that $\sum_{i=1}^{\ell} \sigma_i = \sigma$.
5. For each $i \in [\ell]$, $\mathbf{P}$ and $\mathbf{V}$ set up the claim $((\boldsymbol{p}_i, \sigma_i), (\boldsymbol{w}_i, \boldsymbol{w}'_i), (\boldsymbol{m}_i, \boldsymbol{m}'_i)) \in \mathsf{TIP}^{(\mathcal{C}', \varepsilon')}$.

For proving $\langle \boldsymbol{m}, \boldsymbol{p} \rangle = \sigma$, we analogously define $\mathsf{SplitClaimIP}(\boldsymbol{m}, \boldsymbol{p}, \sigma, (\llbracket \boldsymbol{w}_i \rrbracket)_{i\in[\ell]})$. We note that these subprotocols formalize what was already done for the message $\boldsymbol{m}$ in Fig. 4.

MLIOPs. For clarity, we often phrase the exposition in this section in the language of multilinear polynomial IOPs (MLIOPs), a notion first introduced in [BCFRRZ25]. In an MLIOP, the prover $\mathbf{P}$ can send oracles of the form $\llbracket p(\boldsymbol{X}) \rrbracket$ containing multilinear polynomials $p(\boldsymbol{X})$, and the verifier $\mathbf{V}$ can query these oracles at a specified evaluation point $\boldsymbol{z}$.

We use the MLIOP viewpoint because, as shown in [BMMS25], an MLIOP for $\mathcal{R}$ can be compiled into an IOR from $\mathcal{R}$ to ME. At a high level, this compilation replaces each multilinear polynomial oracle $\llbracket \hat{p}(\boldsymbol{X}) \rrbracket$ with a vector oracle $\llbracket \boldsymbol{w}_{\boldsymbol{p}} \rrbracket$, where $\boldsymbol{w}_{\boldsymbol{p}}$ is the encoding under an appropriate code $\mathcal{C}$ of the evaluation vector $\boldsymbol{p}$ of $\hat{p}(\boldsymbol{X})$ over the boolean hypercube. An evaluation query to $\llbracket \hat{p}(\boldsymbol{X}) \rrbracket$ at a point $\boldsymbol{z}$ then becomes an ME claim about $\llbracket \boldsymbol{w}_{\boldsymbol{p}} \rrbracket$.

The logUp-based permutation check. The logUp protocol (which is the unofficial name for the lookup argument from [Hab22]) yields an efficient 'permutation check' that we make use of in CodeswitchClaims. In a permutation check, $\mathbf{P}$ wants to commit to vectors $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{F}^k$ and convince $\mathbf{V}$ that $\boldsymbol{q}$ is obtained by permuting $\boldsymbol{p}$ with respect to a permutation $\pi : [k] \to [k]$. To commit to $\boldsymbol{p}$ and $\boldsymbol{q}$, $\mathbf{P}$ sends the oracles $\llbracket \hat{p}(\boldsymbol{X}) \rrbracket$ and $\llbracket \hat{q}(\boldsymbol{X}) \rrbracket$ to $\mathbf{V}$. In order to prove the permutation claim, $\mathbf{P}$ and $\mathbf{V}$ define the vectors $\mathbf{id}, \boldsymbol{\pi} \in \mathbb{F}^k$ such that for each $i \in [k]$, $\mathbf{id}[i] := i - 1 \in \mathbb{F}$ and $\boldsymbol{\pi}[j] = \pi(j) - 1 \in \mathbb{F}$, where $i \in \mathbb{F}$ is obtained by adding $1 \in \mathbb{F}$ to itself i times. It is easy to see that $\boldsymbol{q}$ is obtained by permuting $\boldsymbol{p}$ with respect to a permutation π if and only if the following polynomials are equal:

$$\prod_{i\in[k]} (X^2 + X \cdot \boldsymbol{\pi}[i] + \boldsymbol{p}[i]) = \prod_{i\in[k]} (X^2 + X \cdot \mathbf{id}[i] + \boldsymbol{q}[i]) \ .$$

It is shown in [Hab22] that these polynomials are equal if and only if their respective logarithmic derivatives are equal, i.e.,

$$\sum_{i\in[k]} \frac{1}{X^2 + X \cdot \boldsymbol{\pi}[i] + \boldsymbol{p}[i]} = \sum_{i\in[k]} \frac{1}{X^2 + X \cdot \mathbf{id}[i] + \boldsymbol{q}[i]} \ .$$

It is also shown that this equality can be checked (with overwhelming probability) by having the verifier sample $\alpha \xleftarrow{\$} \mathbb{F}$ and checking that

$$\sum_{i\in[k]} \frac{1}{\alpha^2 + \alpha \cdot \boldsymbol{\pi}[i] + \boldsymbol{p}[i]} = \sum_{i\in[k]} \frac{1}{\alpha^2 + \alpha \cdot \mathbf{id}[i] + \boldsymbol{q}[i]} \ .$$

This is essentially a sumcheck claim over *rational polynomials*, which the prover can prove using the fractional sumcheck. In more detail, this can be proved as follows:

- $\mathbf{P}$ defines the 'permutation witness' vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^k$ such that $\boldsymbol{a}[i] := (\alpha^2 + \alpha \cdot \boldsymbol{\pi}[i] + \boldsymbol{p}[i])^{-1}$ and $\boldsymbol{b}[i] := (\alpha^2 + \alpha \cdot \mathbf{id}[i] + \boldsymbol{q}[i])^{-1}$, for all $i \in [k]$, and sends the oracles $\llbracket \hat{a}(\boldsymbol{X}) \rrbracket$ and $\llbracket \hat{b}(\boldsymbol{X}) \rrbracket$ to $\mathbf{V}$.

- $\mathbf{P}$ and $\mathbf{V}$ run the zerocheck protocol to verify that the foregoing equalities hold. That is, $\mathbf{V}$ samples $\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^{\log k}$, and $\mathbf{P}$ and $\mathbf{V}$ run the regular multilinear sumcheck protocol to check that

$$\sum_{\boldsymbol{x}\in\{0,1\}^{\log k}} \left(\hat{a}(\boldsymbol{x}) \cdot \left(\alpha^2 + \alpha \cdot \hat{\pi}(\boldsymbol{x}) + \hat{p}(\boldsymbol{x})\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{x}, \boldsymbol{r}) = 0, \text{ and}$$
$$\sum_{\boldsymbol{x}\in\{0,1\}^{\log k}} \left(\hat{b}(\boldsymbol{x}) \cdot \left(\alpha^2 + \alpha \cdot \hat{\mathsf{id}}(\boldsymbol{x}) + \hat{q}(\boldsymbol{x})\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{x}, \boldsymbol{r}) = 0 \ .$$

- $\mathbf{P}$ and $\mathbf{V}$ run the multilinear sumcheck protocol to check that $\sum_{\boldsymbol{b}\in\{0,1\}^{\log k}} \hat{a}(\boldsymbol{b}) = \sum_{\boldsymbol{b}\in\{0,1\}^{\log k}} \hat{b}(\boldsymbol{b})$.

Remark 6.3 (Evaluating $\hat{\mathsf{id}}(\boldsymbol{X})$ and $\hat{\pi}(\boldsymbol{X})$)**.** At the end of the foregoing zerocheck protocol, $\mathbf{V}$ will need to evaluate $\hat{\mathsf{id}}(\boldsymbol{X})$ and $\hat{\pi}(\boldsymbol{X})$ at a random point $\boldsymbol{r}_{\mathsf{sc}} \xleftarrow{\$} \mathbb{F}^{\log k}$ sampled by $\mathbf{V}$. This can be done as follows:

- $\hat{\mathsf{id}}(\boldsymbol{X})$ can be evaluated by $\mathbf{V}$ in $O(\log k)$ time by exploiting the structured nature of $\mathbf{id}$. In more detail, it is easy to see that the multilinear extension of $\mathbf{id}$ is

$$\hat{\mathsf{id}}(X_1, \ldots, X_{\log k}) = \sum_{i=1}^{\log k} 2^{\log k - i} \cdot X_i \ .$$

  Thus $\hat{\mathsf{id}}(\boldsymbol{r}_{\mathsf{sc}}) = \sum_{i=1}^{\log k} 2^{\log k - i} \cdot \boldsymbol{r}_{\mathsf{sc}}[i]$ can be evaluated by $\mathbf{V}$ in $O(\log k)$ time.
- Since $\boldsymbol{\pi}$ could correspond to any permutation, we assume $\mathbf{V}$ has access to the oracle $[\![\hat{\pi}(\boldsymbol{X})]\!]$, allowing it to (succinctly) evaluate it at $\boldsymbol{r}_{\mathsf{sc}}$. Depending on the application, this oracle is either generated in a preprocessing phase (as in our case), or sent by $\mathbf{P}$ during the interaction.

This completes the overview of the logUp-based permutation check. We refer the reader to [Hab22] for more details.

6.2 High-level overview of CodeswitchOracles and CodeswitchClaims

Before presenting the full constructions of CodeswitchOracles and CodeswitchClaims, we give a high-level overview in the language of MLIOPs. CodeswitchOracles essentially preprocesses and outputs the following multilinear polynomial oracles:

- Given the permutations π_1 and π_2, it defines the permutation vectors $\boldsymbol{\pi}_1, \boldsymbol{\pi}_2 \in \mathbb{F}^{n_{\mathsf{ERA}}}$ such that for each $i \in [n_{\mathsf{ERA}}]$, $\boldsymbol{\pi}_1[i] = \pi_1(i) - 1$ and $\boldsymbol{\pi}_2[i] = \pi_2(i) - 1$, and outputs the oracles $[\![\hat{\pi}_1(\boldsymbol{X})]\!]$ and $[\![\hat{\pi}_2(\boldsymbol{X})]\!]$, and
- given the multiplication vector $\boldsymbol{v}$, it outputs the oracle $[\![\hat{v}(\boldsymbol{X})]\!]$.

At a high level, CodeswitchClaims addresses the following task: $\mathbf{V}$ holds a multilinear oracle $[\![\hat{m}(\boldsymbol{X})]\!]$ corresponding to a vector $\boldsymbol{m} \in \mathbb{F}^k$, and wants to be convinced that $\mathsf{Enc}_{\mathsf{ERA}}(\boldsymbol{m})[\alpha_j] = \sigma_j^{\mathsf{cs}}$ for all $j \in [\nu]$. We denote the intermediate vectors until the end of the first stage by $\boldsymbol{w}^{\mathcal{B}} := \boldsymbol{G}_{\mathcal{B}} \cdot \boldsymbol{m}$, $\boldsymbol{w}^{\mathsf{rep}} := \boldsymbol{R} \cdot \boldsymbol{w}^{\mathcal{B}}$, $\boldsymbol{w}^{\mathsf{perm}_1} := \boldsymbol{P}_1 \cdot \boldsymbol{w}^{\mathsf{rep}}$, $\boldsymbol{w}^{\mathsf{mult}_1} := \boldsymbol{M} \cdot \boldsymbol{w}^{\mathsf{perm}_1}$ and $\boldsymbol{w}^{\mathsf{acc}_1} := \boldsymbol{A} \cdot \boldsymbol{w}^{\mathsf{mult}_1}$, and the rest of the intermediate vectors $\boldsymbol{w}^{\mathsf{perm}_2} := \boldsymbol{P}_2 \cdot \boldsymbol{w}^{\mathsf{acc}_1}$, $\boldsymbol{w}^{\mathsf{mult}_2} := \boldsymbol{M} \cdot \boldsymbol{w}^{\mathsf{perm}_2}$, and $\boldsymbol{w}^{\mathsf{acc}_2} := \boldsymbol{A} \cdot \boldsymbol{w}^{\mathsf{mult}_2}$. In order to prove the codeswitch claims, $\mathbf{P}$ sends $[\![\hat{w}^{\mathcal{B}}(\boldsymbol{X})]\!]$, $[\![\hat{w}^{\mathsf{perm}_1}(\boldsymbol{X})]\!]$ and $[\![\hat{w}^{\mathsf{perm}_2}(\boldsymbol{X})]\!]$ to $\mathbf{V}$.

Validating $[\![\hat{w}^{\mathcal{B}}(\boldsymbol{X})]\!]$. We use the techniques of [BMMS25] to validate that $\boldsymbol{w}^{\mathcal{B}} = \boldsymbol{G}_{\mathcal{B}} \cdot \boldsymbol{m}$, for the vector $\boldsymbol{w}^{\mathcal{B}}$ corresponding to $[\![\hat{w}^{\mathcal{B}}(\boldsymbol{X})]\!]$. We omit the details for brevity, but note that this requires CodeswitchClaims to also output an oracle that contains information regarding $\boldsymbol{G} \in \mathbb{F}^{\sqrt{n_{\mathcal{B}}} \times \sqrt{k}}$, the generator matrix of the code $\mathcal{B}'$ such that $\mathcal{B} = \mathcal{B}' \otimes \mathcal{B}'$.

Emulating $[\![\hat{w}^{\mathsf{rep}}(\boldsymbol{X})]\!]$. We do not ask $\mathbf{P}$ to send $[\![\hat{w}^{\mathsf{rep}}(\boldsymbol{X})]\!]$, because $\mathbf{V}$ can 'emulate' this oracle directly from $[\![\hat{w}^{\mathcal{B}}(\boldsymbol{X})]\!]$. To see this, take any $\boldsymbol{z} \in \mathbb{F}^{\log n_{\mathsf{ERA}}}$ and parse it as $\boldsymbol{z} = (\boldsymbol{z}_1, \boldsymbol{z}_2)$ where $\boldsymbol{z}_1 \in \mathbb{F}^{\log r}$ and $\boldsymbol{z}_2 \in \mathbb{F}^{\log n_{\mathcal{B}}}$. Since $\boldsymbol{w}^{\mathsf{rep}}$ is obtained by repeating $\boldsymbol{w}^{\mathcal{B}}$ r-times, the following holds:

$$\hat{w}^{\mathsf{rep}}(\boldsymbol{z}) = \sum_{\boldsymbol{b} \in \{0,1\}^{\log r}} \mathrm{eq}(\boldsymbol{b}, \boldsymbol{z}_1) \cdot \hat{w}^{\mathcal{B}}(\boldsymbol{z}_2) = \left(\sum_{\boldsymbol{b} \in \{0,1\}^{\log r}} \mathrm{eq}(\boldsymbol{b}, \boldsymbol{z}_1) \right) \cdot \hat{w}^{\mathcal{B}}(\boldsymbol{z}_2) \ .$$

Validating $[\![\hat{w}^{\mathsf{perm}_1}(\boldsymbol{X})]\!]$. Given that we have access to the oracles $[\![\hat{\pi}_1(\boldsymbol{X})]\!]$, $[\![\hat{w}^{\mathsf{rep}}(\boldsymbol{X})]\!]$ (via emulation), and $[\![\hat{w}^{\mathsf{perm}_1}(\boldsymbol{X})]\!]$, we can use the logUp-based permutation check described above to validate that $\boldsymbol{w}^{\mathsf{perm}_1}$ is indeed obtained by permuting $\boldsymbol{w}^{\mathsf{rep}}$ with respect to the permutation π_1.

Emulating $[\![\hat{w}^{\mathsf{acc}_1}(\boldsymbol{X})]\!]$. Once $\mathbf{V}$ has validated $[\![\hat{w}^{\mathsf{perm}_1}(\boldsymbol{X})]\!]$, it can use it to emulate $[\![\hat{w}^{\mathsf{acc}_1}(\boldsymbol{X})]\!]$. Indeed, the reply to a query to $[\![\hat{w}^{\mathsf{acc}_1}(\boldsymbol{X})]\!]$ at a point $\boldsymbol{z}$ is precisely $\langle \boldsymbol{w}^{\mathsf{acc}_1}, \mathbf{eq}(\boldsymbol{z}) \rangle$, which can be rewritten as

$$\begin{aligned}
\langle \boldsymbol{w}^{\mathsf{acc}_1}, \mathbf{eq}(\boldsymbol{z}) \rangle &= \sum_{j \in [n_{\mathsf{ERA}}]} \boldsymbol{w}^{\mathsf{acc}_1}[j] \cdot \mathbf{eq}(\boldsymbol{z})[j] \\
&= \sum_{j \in [n_{\mathsf{ERA}}]} \left(\sum_{j' \in [n_{\mathsf{ERA}}]} \boldsymbol{A}[j, j'] \cdot \boldsymbol{w}^{\mathsf{mult}_1}[j'] \right) \cdot \mathbf{eq}(\boldsymbol{z})[j] \\
&= \sum_{j \in [n_{\mathsf{ERA}}]} \left(\sum_{j' \in [n_{\mathsf{ERA}}]} \boldsymbol{A}[j, j'] \cdot \boldsymbol{w}^{\mathsf{perm}_1}[j'] \cdot \boldsymbol{v}[j'] \right) \cdot \mathbf{eq}(\boldsymbol{z})[j] \\
&= \sum_{j' \in [n_{\mathsf{ERA}}]} \left(\sum_{j \in [n_{\mathsf{ERA}}]} \boldsymbol{A}[j, j'] \cdot \mathbf{eq}(\boldsymbol{z})[j] \right) \cdot \boldsymbol{w}^{\mathsf{perm}_1}[j'] \cdot \boldsymbol{v}[j'] \\
&= \sum_{j' \in [n_{\mathsf{ERA}}]} \boldsymbol{A}_{\mathbf{eq}(\boldsymbol{z})}[j'] \cdot \boldsymbol{w}^{\mathsf{perm}_1}[j'] \cdot \boldsymbol{v}[j'] \\
&= \langle \boldsymbol{A}_{\mathbf{eq}(\boldsymbol{z})}, \boldsymbol{w}^{\mathsf{perm}_1}, \boldsymbol{v} \rangle \ ,
\end{aligned}$$

where $\boldsymbol{A}_{\mathbf{eq}(\boldsymbol{z})}$ is the vector obtained by taking the linear combination of the rows of $\boldsymbol{A}$ according to the coefficients in $\mathbf{eq}(\boldsymbol{z})$. This can be expressed as a sumcheck claim

$$\langle \boldsymbol{w}^{\mathsf{acc}_1}, \mathbf{eq}(\boldsymbol{z}) \rangle = \sum_{\boldsymbol{b} \in \{0,1\}^{\log n_{\mathsf{ERA}}}} \hat{A}_{\mathbf{eq}(\boldsymbol{z})}(\boldsymbol{b}) \cdot \hat{w}^{\mathsf{perm}_1}(\boldsymbol{b}) \cdot \hat{v}(\boldsymbol{b}) \ ,$$

which can be reduced to evaluation claims about $\hat{A}_{\mathbf{eq}(\boldsymbol{z})}(\boldsymbol{X})$, $\hat{w}^{\mathsf{perm}_1}(\boldsymbol{X})$ and $\hat{v}(\boldsymbol{X})$ at a random point via the sumcheck protocol. It is shown in [BCFRRZ25, Proposition 5.3] that by exploiting the structured and sparse nature of $\boldsymbol{A}$, $\mathbf{P}$ can compute $\boldsymbol{A}_{\mathbf{eq}(\boldsymbol{r}^{\mathsf{perm},2})}$ in $O(n_{\mathsf{ERA}})$ time and $\mathbf{V}$ can evaluate $\hat{A}_{\mathbf{eq}(\boldsymbol{z})}(\boldsymbol{X})$ in $O(\log n_{\mathsf{ERA}})$ time, allowing $\mathbf{P}$ and $\mathbf{V}$ to run the foregoing sumcheck protocol efficiently. Thus, queries to $[\![\hat{w}^{\mathsf{acc}_1}(\boldsymbol{X})]\!]$ reduce to queries to $[\![\hat{w}^{\mathsf{perm}_1}(\boldsymbol{X})]\!]$ and $[\![\hat{v}(\boldsymbol{X})]\!]$.

Emulating $[\![\hat{w}^{\mathsf{acc}_2}(\boldsymbol{X})]\!]$. The same ideas apply to the second stage:

- first validate $[\![\hat{w}^{\mathsf{perm}_2}(\boldsymbol{X})]\!]$ using the logUp-based permutation check with respect to $[\![\hat{w}^{\mathsf{acc}_1}(\boldsymbol{X})]\!]$ and $[\![\hat{\pi}_2(\boldsymbol{X})]\!]$, and then
- use the validated $[\![\hat{w}^{\mathsf{perm}_2}(\boldsymbol{X})]\!]$ to emulate $[\![\hat{w}^{\mathsf{acc}_2}(\boldsymbol{X})]\!]$, by reducing queries to $[\![\hat{w}^{\mathsf{acc}_2}(\boldsymbol{X})]\!]$ to queries to $[\![\hat{w}^{\mathsf{perm}_2}(\boldsymbol{X})]\!]$ and $[\![\hat{v}(\boldsymbol{X})]\!]$.

$\mathbf{V}$ now has the ability to emulate queries to $[\![\hat{w}^{\mathsf{acc}_2}(\boldsymbol{X})]\!]$, where it has validated that $\boldsymbol{w}^{\mathsf{acc}_2} = \mathsf{Enc}_{\mathsf{ERA}}(\boldsymbol{m})$.

Overall, this reduces checking $\mathsf{Enc}_{\mathsf{ERA}}(\boldsymbol{m})[\alpha_j] = \sigma_j^{\mathsf{cs}}$ for all $j \in [\nu]$ to a single sumcheck claim about $\hat{w}^{\mathsf{acc}_2}(\boldsymbol{X})$, which in turn reduces to a single query to $[\![\hat{w}^{\mathsf{acc}_2}(\boldsymbol{X})]\!]$ via the sumcheck protocol. This completes the high-level overview.

6.3 Full constructions of CodeswitchOracles and CodeswitchClaims

We present the helper function CodeswitchOracles in Fig. 5 and the subprotocol CodeswitchClaims in Fig. 6. While we presented the overview in the language of MLIOPs, in this section we directly present an IOR, using SplitAndEncode, SplitClaimIP and SplitClaimTIP, in order to ensure we output a single claim about $\mathcal{C}'$.

In the interest of clarity, we will not be explicit about $\mathsf{numIP}, \mathsf{numTIP}, \ell_{\mathsf{aux}}$, or about denoting the auxiliary oracles $([\![\boldsymbol{w}_i^{\mathsf{aux}}]\!])_{i\in[\ell_{\mathsf{aux}}]}$ in the protocol description, but will instead analyze these parameters in Section 6.5.

CodeswitchOracles(ERA) :

1. Let $\boldsymbol{G} \in \mathbb{F}^{\sqrt{n_{\mathcal{B}}}\times\sqrt{k}}$ be the generator matrix of the code $\mathcal{B}'$, such that $\mathcal{B} = \mathcal{B}' \otimes \mathcal{B}'$.
2. Define $\boldsymbol{g} \in \mathbb{F}^{\sqrt{n_{\mathcal{B}}k}}$ as the concatenation of the rows of $\boldsymbol{G}$, and define $\ell_{\boldsymbol{g}} := \lceil \sqrt{n_{\mathcal{B}}k}/k' \rceil$.
3. Compute $([\boldsymbol{g}_i]_{i\in[\ell_{\boldsymbol{g}}]}, [\boldsymbol{w}_i^{\boldsymbol{g}}]_{i\in[\ell_{\boldsymbol{g}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{g})$.
4. For each $s \in [2]$:
   - Define $\boldsymbol{\pi}_s \in \mathbb{F}^{n_{\mathsf{ERA}}}$ such that for each $s \in [n_{\mathsf{ERA}}]$, $\boldsymbol{\pi}_s[j] = \pi_s(j) - 1$.
   - Compute $([\boldsymbol{\pi}_{s,i}]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{w}_i^{\boldsymbol{\pi}_s}]_{i\in[\ell_{\mathsf{ERA}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{\pi}_s)$.
5. Compute $([\boldsymbol{v}_i]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{w}_i^{\boldsymbol{v}}]_{i\in[\ell_{\mathsf{ERA}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{v})$.
6. Output $\big([\boldsymbol{w}_i^{\boldsymbol{g}}]_{i\in[\ell_{\boldsymbol{g}}]}, [\boldsymbol{w}_i^{\boldsymbol{\pi}_1}]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{w}_i^{\boldsymbol{\pi}_2}]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{w}_i^{\boldsymbol{v}}]_{i\in[\ell_{\mathsf{ERA}}]}\big)$.

Figure 5: The helper function CodeswitchOracles.

CodeswitchClaims :

Goal: To prove that for all $j \in [\nu]$, $\mathsf{Enc}_{\mathsf{ERA}}(\boldsymbol{m})[\alpha_j] = \sigma_j^{\mathsf{cs}}$.

Protocol:

1. $\mathbf{P}$ computes the intermediate words $\boldsymbol{w}^{\mathcal{B}} := \mathsf{Enc}_{\mathcal{B}}(\boldsymbol{m})$, $\boldsymbol{w}^{\mathsf{rep}} := \boldsymbol{R} \cdot \boldsymbol{w}^{\mathcal{B}}$, $\boldsymbol{w}^{\mathsf{perm}_1} := \boldsymbol{P}_1 \cdot \boldsymbol{w}^{\mathsf{rep}}$, $\boldsymbol{w}^{\mathsf{mult}_1} := \boldsymbol{M} \cdot \boldsymbol{w}^{\mathsf{perm}_1}$ and $\boldsymbol{w}^{\mathsf{acc}_1} := \boldsymbol{A} \cdot \boldsymbol{w}^{\mathsf{mult}_1}$.

   Checking the base code encoding.
2. $\mathbf{P}$ computes $([\boldsymbol{w}_i^{\mathcal{B}}]_{i\in[\ell_{\mathcal{B}}]}, [\mathbf{o}_i^{\mathcal{B}}]_{i\in[\ell_{\mathcal{B}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{w}^{\mathcal{B}})$ where $\ell_{\mathcal{B}} := n_{\mathcal{B}}/k'$.
3. $\mathbf{P}$ sends the oracles $([\![\mathbf{o}_i^{\mathcal{B}}]\!])_{i\in[\ell_{\mathcal{B}}]}$ to $\mathbf{V}$, and an out-of-domain sample is performed:
   - $\mathbf{V}$ samples $\boldsymbol{z}_{\mathsf{ood}}^{\mathcal{B}} \xleftarrow{\$} \mathbb{F}^{\log n_{\mathcal{B}}}$ and sends it to $\mathbf{P}$.
   - $\mathbf{P}$ sends $\sigma_{\mathsf{ood}}^{\mathcal{B}} := \hat{w}^{\mathcal{B}}(\boldsymbol{z}_{\mathsf{ood}}^{\mathcal{B}})$ to $\mathbf{V}$.
   - $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\mathcal{B}}, \mathbf{eq}(\boldsymbol{z}_{\mathsf{ood}}^{\mathcal{B}}), \sigma_{\mathsf{ood}}^{\mathcal{B}}, ([\![\mathbf{o}_i^{\mathcal{B}}]\!])_{i\in[\ell_{\mathcal{B}}]})$.
4. $\mathbf{P}$ wants to convince $\mathbf{V}$ that the following holds for all $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2) \in \{0,1\}^{\log n_{\mathcal{B}}}$:

$$\sum_{\boldsymbol{y}=(\boldsymbol{y}_1,\boldsymbol{y}_2)\in\{0,1\}^{\log k}} \hat{g}(\boldsymbol{x}_1,\boldsymbol{y}_1)\cdot\hat{g}(\boldsymbol{x}_2,\boldsymbol{y}_2)\cdot\hat{m}(\boldsymbol{y}) = \hat{w}^{\mathcal{B}}(\boldsymbol{x}) \ ,$$

where $\hat{g}(\boldsymbol{X}_1, \boldsymbol{Y}_1)$ is the multilinear extension of the generator matrix of the $\mathcal{B}'$ such that $\mathcal{B} = \mathcal{B}' \otimes \mathcal{B}'$, $\boldsymbol{x}_1, \boldsymbol{x}_2 \in (\log n_{\mathcal{B}})/2$ and $\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{F}^{(\log k)/2}$.

5. Since the polynomials on both sides are multilinear in $\boldsymbol{x}$, it suffices to check their equality at a random point. Thus $\mathbf{V}$ samples $\boldsymbol{r}^{\boldsymbol{x}} \xleftarrow{\$} \mathbb{F}^{\log n_{\mathcal{B}}}$ and sends it to $\mathbf{P}$.
6. $\mathbf{P}$ sends $\sigma^{\mathcal{B}}_{\mathsf{sc}} := \hat{w}^{\mathcal{B}}(\boldsymbol{r}^{\boldsymbol{x}})$ to $\mathbf{V}$.
7. $\mathbf{P}$ and $\mathbf{V}$ parse $\boldsymbol{r}^{\boldsymbol{x}} = (\boldsymbol{r}^{\boldsymbol{x}}_1, \boldsymbol{r}^{\boldsymbol{x}}_2)$ and run sumcheck to reduce checking the claim

$$\sum_{\boldsymbol{y}=(\boldsymbol{y}_1,\boldsymbol{y}_2)\in\{0,1\}^{\log k}} \hat{g}(\boldsymbol{r}^{\boldsymbol{x}}_1, \boldsymbol{y}_1) \cdot \hat{g}(\boldsymbol{r}^{\boldsymbol{x}}_2, \boldsymbol{y}_2) \cdot \hat{m}(\boldsymbol{y}) = \sigma^{\mathcal{B}}_{\mathsf{sc}}$$

to checking the following claim at a random point $\boldsymbol{r}^{\boldsymbol{y}} = (\boldsymbol{r}^{\boldsymbol{y}}_1, \boldsymbol{r}^{\boldsymbol{y}}_2)$ sampled by $\mathbf{V}$:

$$\hat{g}(\boldsymbol{r}^{\boldsymbol{x}}_1, \boldsymbol{r}^{\boldsymbol{y}}_1) \cdot \hat{g}(\boldsymbol{r}^{\boldsymbol{x}}_2, \boldsymbol{r}^{\boldsymbol{y}}_2) \cdot \hat{m}(\boldsymbol{r}^{\boldsymbol{y}}) = \sigma^{\mathcal{B}}_{\mathsf{eval}} \ .$$

8. $\mathbf{P}$ sends $\sigma^{\mathcal{B}}_{\boldsymbol{g},1} := \hat{g}(\boldsymbol{r}^{\boldsymbol{x}}_1, \boldsymbol{r}^{\boldsymbol{y}}_1)$, $\sigma^{\mathcal{B}}_{\boldsymbol{g},2} := \hat{g}(\boldsymbol{r}^{\boldsymbol{x}}_2, \boldsymbol{r}^{\boldsymbol{y}}_2)$ and $\sigma^{\mathcal{B}}_{\boldsymbol{m}} := \hat{m}(\boldsymbol{r}^{\boldsymbol{y}})$ to $\mathbf{V}$, who checks that

$$\sigma^{\mathcal{B}}_{\boldsymbol{g},1} \cdot \sigma^{\mathcal{B}}_{\boldsymbol{g},2} \cdot \sigma^{\mathcal{B}}_{\boldsymbol{m}} = \sigma^{\mathcal{B}}_{\mathsf{eval}} \ .$$

9. $\mathbf{P}$ and $\mathbf{V}$ run
   - $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\mathcal{B}}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{x}}), \sigma^{\mathcal{B}}_{\mathsf{sc}}, (\llbracket \mathbf{o}^{\mathcal{B}}_i \rrbracket)_{i\in[\ell_{\mathcal{B}}]})$,
   - $\mathsf{SplitClaimIP}(\boldsymbol{g}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{x}}_1, \boldsymbol{r}^{\boldsymbol{y}}_1), \sigma^{\mathcal{B}}_{\boldsymbol{g},1}, (\llbracket \boldsymbol{w}^{\boldsymbol{g}}_i \rrbracket)_{i\in[\ell_{\boldsymbol{g}}]})$,
   - $\mathsf{SplitClaimIP}(\boldsymbol{g}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{x}}_2, \boldsymbol{r}^{\boldsymbol{y}}_2), \sigma^{\mathcal{B}}_{\boldsymbol{g},2}, (\llbracket \boldsymbol{w}^{\boldsymbol{g}}_i \rrbracket)_{i\in[\ell_{\boldsymbol{g}}]})$, and
   - $\mathsf{SplitClaimIP}(\boldsymbol{m}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{y}}), \sigma^{\mathcal{B}}_{\boldsymbol{m}}, (\llbracket \boldsymbol{w}_i \rrbracket)_{i\in[\ell_{\boldsymbol{m}}]})$.

Emulating the repeat step.

10. Define $([\boldsymbol{w}^{\mathsf{rep}}_i]_{i\in[\ell_{\mathsf{ERA}}]}, [\mathbf{o}^{\mathsf{rep}}_i]_{i\in[\ell_{\mathsf{ERA}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{w}^{\mathsf{rep}})$.
11. $\mathbf{V}$ simulates the oracles $(\llbracket \mathbf{o}^{\mathsf{rep}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]}$ by repeating the oracles $(\llbracket \mathbf{o}^{\mathcal{B}}_i \rrbracket)_{i\in[\ell_{\mathcal{B}}]}$ r times.

Checking the first permute step.

12. $\mathbf{P}$ computes $([\boldsymbol{w}^{\mathsf{perm}_1}_i]_{i\in[\ell_{\mathsf{ERA}}]}, [\mathbf{o}^{\mathsf{perm}_1}_i]_{i\in[\ell_{\mathsf{ERA}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{w}^{\mathsf{perm}_1})$.
13. $\mathbf{P}$ sends the oracles $(\llbracket \mathbf{o}^{\mathsf{perm}_1}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]}$ to $\mathbf{V}$, and an out-of-domain sample is performed:
    - $\mathbf{V}$ samples $\boldsymbol{z}^{\mathsf{perm}_1}_{\mathsf{ood}} \xleftarrow{\$} \mathbb{F}^{\log n_{\mathsf{ERA}}}$ and sends it to $\mathbf{P}$.
    - $\mathbf{P}$ sends $\sigma^{\mathsf{perm}_1}_{\mathsf{ood}} := \hat{w}^{\mathsf{perm}_1}(\boldsymbol{z}^{\mathsf{perm}_1}_{\mathsf{ood}})$ to $\mathbf{V}$.
    - $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\mathsf{perm}_1}, \mathbf{eq}(\boldsymbol{z}^{\mathsf{perm}_1}_{\mathsf{ood}}), \sigma^{\mathsf{perm}_1}_{\mathsf{ood}}, (\llbracket \mathbf{o}^{\mathsf{perm}_1}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.
14. $\mathbf{P}$ must now convince $\mathbf{V}$ that $\boldsymbol{w}^{\mathsf{perm}_1}$ is indeed the permuted version of $\boldsymbol{w}^{\mathsf{rep}}$, which is done via the logUp-based permutation check as follows:
    - $\mathbf{V}$ samples a random challenge $\alpha \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathbf{P}$.
    - $\mathbf{P}$ defines the permutation witness vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^{n_{\mathsf{ERA}}}$ such that $\boldsymbol{a}[j] := (\alpha^2 + \alpha \cdot \boldsymbol{\pi}_1[j] + \boldsymbol{w}^{\mathsf{rep}}[j])^{-1}$ and $\boldsymbol{b}[j] := (\alpha^2 + \alpha \cdot \mathbf{id}[j] + \boldsymbol{w}^{\mathsf{perm}_1}[j])^{-1}$, for all $j \in [n_{\mathsf{ERA}}]$.
    - $\mathbf{P}$ computes $([\boldsymbol{a}_i]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{w}^{\boldsymbol{a}}_i]_{i\in[\ell_{\mathsf{ERA}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{a})$ and $([\boldsymbol{b}_i]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{w}^{\boldsymbol{b}}_i]_{i\in[\ell_{\mathsf{ERA}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{b})$.
    - $\mathbf{P}$ sends the oracles $(\llbracket \boldsymbol{w}^{\boldsymbol{a}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]}$ to $\mathbf{V}$, and an out-of-domain sample is performed:
      - $\mathbf{V}$ samples $\boldsymbol{z}^{\boldsymbol{a}}_{\mathsf{ood}} \xleftarrow{\$} \mathbb{F}^{\log n_{\mathsf{ERA}}}$ and sends it to $\mathbf{P}$.
      - $\mathbf{P}$ sends $\sigma^{\boldsymbol{a}}_{\mathsf{ood}} := \hat{w}^{\boldsymbol{a}}(\boldsymbol{z}^{\boldsymbol{a}}_{\mathsf{ood}})$ to $\mathbf{V}$.

  - $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\boldsymbol{a}}, \mathbf{eq}(\boldsymbol{z}^{\boldsymbol{a}}_{\mathsf{ood}}), \sigma^{\boldsymbol{a}}_{\mathsf{ood}}, (\llbracket \boldsymbol{w}^{\boldsymbol{a}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.
- $\mathbf{P}$ sends the oracles $(\llbracket \boldsymbol{w}^{\boldsymbol{b}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]}$ to $\mathbf{V}$, and an out-of-domain sample is performed:
  - $\mathbf{V}$ samples $\boldsymbol{z}^{\boldsymbol{b}}_{\mathsf{ood}} \xleftarrow{\$} \mathbb{F}^{\log n_{\mathsf{ERA}}}$ and sends it to $\mathbf{P}$.
  - $\mathbf{P}$ sends $\sigma^{\boldsymbol{b}}_{\mathsf{ood}} := \hat{w}^{\boldsymbol{b}}(\boldsymbol{z}^{\boldsymbol{b}}_{\mathsf{ood}})$ to $\mathbf{V}$.
  - $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\boldsymbol{b}}, \mathbf{eq}(\boldsymbol{z}^{\boldsymbol{b}}_{\mathsf{ood}}), \sigma^{\boldsymbol{b}}_{\mathsf{ood}}, (\llbracket \boldsymbol{w}^{\boldsymbol{b}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.
- $\mathbf{P}$ and $\mathbf{V}$ run zerochecks to check that $\boldsymbol{a}[j] \cdot (\alpha^2 + \alpha \cdot \boldsymbol{\pi}_1[j] + \boldsymbol{w}^{\mathsf{rep}}[j]) = 1$ and $\boldsymbol{b}[j] \cdot (\alpha^2 + \alpha \cdot \mathbf{id}[j] + \boldsymbol{w}^{\mathsf{perm}_1}[j]) = 1$ for all $j \in [n_{\mathsf{ERA}}]$ by doing the following:
  - $\mathbf{V}$ samples $\boldsymbol{r}^{\mathsf{perm}_1} \xleftarrow{\$} \mathbb{F}^{\log n_{\mathsf{ERA}}}$ and sends it to $\mathbf{P}$.
  - $\mathbf{P}$ and $\mathbf{V}$ run sumcheck to reduce checking the claim

$$\sum_{\boldsymbol{x}\in\{0,1\}^{\log n_{\mathsf{ERA}}}} \left(\hat{a}(\boldsymbol{x}) \cdot \left(\alpha^2 + \alpha \cdot \hat{\pi}_1(\boldsymbol{x}) + \hat{w}^{\mathsf{rep}}(\boldsymbol{x})\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{x}, \boldsymbol{r}^{\mathsf{perm}_1}) = 0$$

    to checking the following claim at a random point $\boldsymbol{r}^{\boldsymbol{a}}$ sampled by $\mathbf{V}$:

$$\left(\hat{a}(\boldsymbol{r}^{\boldsymbol{a}}) \cdot \left(\alpha^2 + \alpha \cdot \hat{\pi}_1(\boldsymbol{r}^{\boldsymbol{a}}) + \hat{w}^{\mathsf{rep}}(\boldsymbol{r}^{\boldsymbol{a}})\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{r}^{\boldsymbol{a}}, \boldsymbol{r}^{\mathsf{perm}_1}) = \sigma^{\boldsymbol{a}}_{\mathsf{eval}} \ .$$

  - $\mathbf{P}$ sends $\sigma^{\mathsf{perm}_1}_{\boldsymbol{a}} := \hat{a}(\boldsymbol{r}^{\boldsymbol{a}})$, $\sigma^{\mathsf{perm}_1}_{\boldsymbol{\pi}_1} := \hat{\pi}_1(\boldsymbol{r}^{\boldsymbol{a}})$ and $\sigma^{\mathsf{perm}_1}_{\mathsf{rep}} := \hat{w}^{\mathsf{rep}}(\boldsymbol{r}^{\boldsymbol{a}})$ to $\mathbf{V}$, who checks that

$$\left(\sigma^{\mathsf{perm}_1}_{\boldsymbol{a}} \cdot \left(\alpha^2 + \alpha \cdot \sigma^{\mathsf{perm}_1}_{\boldsymbol{\pi}_1} + \sigma^{\mathsf{perm}_1}_{\mathsf{rep}}\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{r}^{\boldsymbol{a}}, \boldsymbol{r}^{\mathsf{perm}_1}) = \sigma^{\boldsymbol{a}}_{\mathsf{eval}} \ .$$

  - $\mathbf{P}$ and $\mathbf{V}$ run sumcheck to reduce checking the claim

$$\sum_{\boldsymbol{x}\in\{0,1\}^{\log n_{\mathsf{ERA}}}} \left(\hat{b}(\boldsymbol{x}) \cdot \left(\alpha^2 + \alpha \cdot \hat{\mathsf{id}}(\boldsymbol{x}) + \hat{w}^{\mathsf{perm}_1}(\boldsymbol{x})\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{x}, \boldsymbol{r}^{\mathsf{perm}_1}) = 0$$

    to checking the following claim at a random point $\boldsymbol{r}^{\boldsymbol{b}}$ sampled by $\mathbf{V}$:

$$\left(\hat{b}(\boldsymbol{r}^{\boldsymbol{b}}) \cdot \left(\alpha^2 + \alpha \cdot \hat{\mathsf{id}}(\boldsymbol{r}^{\boldsymbol{b}}) + \hat{w}^{\mathsf{perm}_1}(\boldsymbol{r}^{\boldsymbol{b}})\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{r}^{\boldsymbol{b}}, \boldsymbol{r}^{\mathsf{perm}_1}) = \sigma^{\boldsymbol{b}}_{\mathsf{eval}} \ .$$

  - $\mathbf{P}$ sends $\sigma^{\mathsf{perm}_1}_{\boldsymbol{b}} := \hat{b}(\boldsymbol{r}^{\boldsymbol{b}})$ and $\sigma^{\mathsf{perm}_1}_{\mathsf{perm}_1} := \hat{w}^{\mathsf{perm}_1}(\boldsymbol{r}^{\boldsymbol{b}})$ to $\mathbf{V}$, who checks that

$$\left(\sigma^{\mathsf{perm}_1}_{\boldsymbol{b}} \cdot \left(\alpha^2 + \alpha \cdot \hat{\mathsf{id}}(\boldsymbol{r}^{\boldsymbol{b}}) + \sigma^{\mathsf{perm}_1}_{\mathsf{perm}_1}\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{r}^{\boldsymbol{b}}, \boldsymbol{r}^{\mathsf{perm}_1}) = \sigma^{\boldsymbol{b}}_{\mathsf{eval}} \ .$$

- $\mathbf{P}$ and $\mathbf{V}$ set up inner product claims corresponding to the zerocheck evaluation claims:
  - $\mathsf{SplitClaimIP}(\boldsymbol{a}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{a}}), \sigma^{\mathsf{perm}_1}_{\boldsymbol{a}}, (\llbracket \boldsymbol{w}^{\boldsymbol{a}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$,
  - $\mathsf{SplitClaimIP}(\boldsymbol{\pi}_1, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{a}}), \sigma^{\mathsf{perm}_1}_{\boldsymbol{\pi}_1}, (\llbracket \boldsymbol{w}^{\boldsymbol{\pi}_1}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$,
  - $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\mathsf{rep}}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{a}}), \sigma^{\mathsf{perm}_1}_{\mathsf{rep}}, (\llbracket \mathbf{o}^{\mathsf{rep}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$,
  - $\mathsf{SplitClaimIP}(\boldsymbol{b}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{b}}), \sigma^{\mathsf{perm}_1}_{\boldsymbol{b}}, (\llbracket \boldsymbol{w}^{\boldsymbol{b}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$, and
  - $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\mathsf{perm}_1}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{b}}), \sigma^{\mathsf{perm}_1}_{\mathsf{perm}_1}, (\llbracket \mathbf{o}^{\mathsf{perm}_1}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.
- $\mathbf{P}$ and $\mathbf{V}$ set up inner product claims that correspond to a zerocheck that enforces that $\boldsymbol{a}[j] = \boldsymbol{b}[j]$, for all $j \in [n_{\mathsf{ERA}}]$ by doing the following:
  - $\mathbf{P}$ sends $\sigma^{\mathsf{perm}_1}_{\boldsymbol{a},\boldsymbol{b}} := \hat{a}(\boldsymbol{r}^{\mathsf{perm}_1})$ to $\mathbf{V}$.
  - $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{a}, \mathbf{eq}(\boldsymbol{r}^{\mathsf{perm}_1}), \sigma^{\mathsf{perm}_1}_{\boldsymbol{a},\boldsymbol{b}}, (\llbracket \boldsymbol{w}^{\boldsymbol{a}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.
  - $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{b}, \mathbf{eq}(\boldsymbol{r}^{\mathsf{perm}_1}), \sigma^{\mathsf{perm}_1}_{\boldsymbol{a},\boldsymbol{b}}, (\llbracket \boldsymbol{w}^{\boldsymbol{b}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.

Checking the second permute step.

15. $\mathbf{P}$ computes the intermediate words $\boldsymbol{w}^{\mathsf{perm}_2} := \boldsymbol{P}_2 \cdot \boldsymbol{w}^{\mathsf{acc}_1}$, $\boldsymbol{w}^{\mathsf{mult}_2} := \boldsymbol{M} \cdot \boldsymbol{w}^{\mathsf{perm}_2}$ and $\boldsymbol{w}^{\mathsf{acc}_2} := \boldsymbol{A} \cdot \boldsymbol{w}^{\mathsf{mult}_2}$ for the second stage.
16. $\mathbf{P}$ computes $([\boldsymbol{w}_i^{\mathsf{perm}_2}]_{i\in[\ell_{\mathsf{ERA}}]}, [\mathbf{o}_i^{\mathsf{perm}_2}]_{i\in[\ell_{\mathsf{ERA}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{w}^{\mathsf{perm}_2})$.
17. $\mathbf{P}$ sends the oracles $(\llbracket \mathbf{o}_i^{\mathsf{perm}_2} \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]}$ to $\mathbf{V}$, and an out-of-domain sample is performed:
    - $\mathbf{V}$ samples $\boldsymbol{z}_{\mathsf{ood}}^{\mathsf{perm}_2} \xleftarrow{\$} \mathbb{F}^{\log n_{\mathsf{ERA}}}$ and sends it to $\mathbf{P}$.
    - $\mathbf{P}$ sends $\sigma_{\mathsf{ood}}^{\mathsf{perm}_2} := \hat{w}^{\mathsf{perm}_2}(\boldsymbol{z}_{\mathsf{ood}}^{\mathsf{perm}_2})$ to $\mathbf{V}$.
    - $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\mathsf{perm}_2}, \mathbf{eq}(\boldsymbol{z}_{\mathsf{ood}}^{\mathsf{perm}_2}), \sigma_{\mathsf{ood}}^{\mathsf{perm}_2}, (\llbracket \mathbf{o}_i^{\mathsf{perm}_2} \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.
18. $\mathbf{P}$ must now convince $\mathbf{V}$ that $\boldsymbol{w}^{\mathsf{perm}_2}$ is indeed the permuted version of $\boldsymbol{w}^{\mathsf{acc}_1}$, which is done in a manner analogous to the previous permutation check, with the modification that instead of directly setting up an evaluation claim about $\hat{w}^{\mathsf{acc}_1}(\boldsymbol{X})$ at the end of the check, $\mathbf{P}$ and $\mathbf{V}$ must emulate this evaluation claim. In more detail, $\mathbf{P}$ and $\mathbf{V}$ do the following:
    - $\mathbf{V}$ samples a random challenge $\beta \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathbf{P}$.
    - $\mathbf{P}$ defines the permutation witness vectors $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{F}^{n_{\mathsf{ERA}}}$ such that $\boldsymbol{c}[j] := (\beta^2 + \beta \cdot \boldsymbol{\pi}_2[j] + \boldsymbol{w}^{\mathsf{acc}_1}[j])^{-1}$ and $\boldsymbol{d}[j] := (\beta^2 + \beta \cdot \mathbf{id}[j] + \boldsymbol{w}^{\mathsf{perm}_2}[j])^{-1}$, for all $j \in [n_{\mathsf{ERA}}]$.
    - $\mathbf{P}$ computes $([\boldsymbol{c}_i]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{w}_i^{\boldsymbol{c}}]_{i\in[\ell_{\mathsf{ERA}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{c})$ and $([\boldsymbol{d}_i]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{w}_i^{\boldsymbol{d}}]_{i\in[\ell_{\mathsf{ERA}}]}) \leftarrow \mathsf{SplitAndEncode}(\boldsymbol{d})$.
    - $\mathbf{P}$ sends the oracles $(\llbracket \boldsymbol{w}_i^{\boldsymbol{c}} \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]}$ to $\mathbf{V}$, and an out-of-domain sample is performed:
        - $\mathbf{V}$ samples $\boldsymbol{z}_{\mathsf{ood}}^{\boldsymbol{c}} \xleftarrow{\$} \mathbb{F}^{\log n_{\mathsf{ERA}}}$ and sends it to $\mathbf{P}$.
        - $\mathbf{P}$ sends $\sigma_{\mathsf{ood}}^{\boldsymbol{c}} := \hat{w}^{\boldsymbol{c}}(\boldsymbol{z}_{\mathsf{ood}}^{\boldsymbol{c}})$ to $\mathbf{V}$.
        - $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\boldsymbol{c}}, \mathbf{eq}(\boldsymbol{z}_{\mathsf{ood}}^{\boldsymbol{c}}), \sigma_{\mathsf{ood}}^{\boldsymbol{c}}, (\llbracket \boldsymbol{w}_i^{\boldsymbol{c}} \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.
    - $\mathbf{P}$ sends the oracles $(\llbracket \boldsymbol{w}_i^{\boldsymbol{d}} \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]}$ to $\mathbf{V}$, and an out-of-domain sample is performed:
        - $\mathbf{V}$ samples $\boldsymbol{z}_{\mathsf{ood}}^{\boldsymbol{d}} \xleftarrow{\$} \mathbb{F}^{\log n_{\mathsf{ERA}}}$ and sends it to $\mathbf{P}$.
        - $\mathbf{P}$ sends $\sigma_{\mathsf{ood}}^{\boldsymbol{d}} := \hat{w}^{\boldsymbol{d}}(\boldsymbol{z}_{\mathsf{ood}}^{\boldsymbol{d}})$ to $\mathbf{V}$.
        - $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\boldsymbol{d}}, \mathbf{eq}(\boldsymbol{z}_{\mathsf{ood}}^{\boldsymbol{d}}), \sigma_{\mathsf{ood}}^{\boldsymbol{d}}, (\llbracket \boldsymbol{w}_i^{\boldsymbol{d}} \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.
    - $\mathbf{P}$ and $\mathbf{V}$ run zerochecks to check that $\boldsymbol{c}[j] \cdot (\beta^2 + \beta \cdot \boldsymbol{\pi}_2[j] + \boldsymbol{w}^{\mathsf{acc}_1}[j]) = 1$ and $\boldsymbol{d}[j] \cdot (\beta^2 + \beta \cdot \mathbf{id}[j] + \boldsymbol{w}^{\mathsf{perm}_2}[j]) = 1$ for all $j \in [n_{\mathsf{ERA}}]$ by doing the following:
        - $\mathbf{V}$ samples $\boldsymbol{r}^{\mathsf{perm}_2} \xleftarrow{\$} \mathbb{F}^{\log n_{\mathsf{ERA}}}$ and sends it to $\mathbf{P}$.
        - $\mathbf{P}$ and $\mathbf{V}$ run sumcheck to reduce checking the claim

$$\sum_{\boldsymbol{x}\in\{0,1\}^{\log n_{\mathsf{ERA}}}} \left(\hat{c}(\boldsymbol{x}) \cdot \left(\beta^2 + \beta \cdot \hat{\pi}_2(\boldsymbol{x}) + \hat{w}^{\mathsf{acc}_1}(\boldsymbol{x})\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{x}, \boldsymbol{r}^{\mathsf{perm}_2}) = 0$$

          to checking the following claim at a random point $\boldsymbol{r}^{\boldsymbol{c}}$ sampled by $\mathbf{V}$:

$$\left(\hat{c}(\boldsymbol{r}^{\boldsymbol{c}}) \cdot \left(\beta^2 + \beta \cdot \hat{\pi}_2(\boldsymbol{r}^{\boldsymbol{c}}) + \hat{w}^{\mathsf{acc}_1}(\boldsymbol{r}^{\boldsymbol{c}})\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{r}^{\boldsymbol{c}}, \boldsymbol{r}^{\mathsf{perm}_2}) = \sigma_{\mathsf{eval}}^{\boldsymbol{c}} \ .$$

        - $\mathbf{P}$ sends $\sigma_{\boldsymbol{c}}^{\mathsf{perm}_2} := \hat{c}(\boldsymbol{r}^{\boldsymbol{c}})$, $\sigma_{\boldsymbol{\pi}_2}^{\mathsf{perm}_2} := \hat{\pi}_2(\boldsymbol{r}^{\boldsymbol{c}})$ and $\sigma_{\mathsf{acc}_1}^{\mathsf{perm}_2} := \hat{w}^{\mathsf{acc}_1}(\boldsymbol{r}^{\boldsymbol{c}})$ to $\mathbf{V}$, who checks that

$$\left(\sigma_{\boldsymbol{c}}^{\mathsf{perm}_2} \cdot \left(\beta^2 + \beta \cdot \sigma_{\boldsymbol{\pi}_2}^{\mathsf{perm}_2} + \sigma_{\mathsf{acc}_1}^{\mathsf{perm}_2}\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{r}^{\boldsymbol{c}}, \boldsymbol{r}^{\mathsf{perm}_2}) = \sigma_{\mathsf{eval}}^{\boldsymbol{c}} \ .$$

        - $\mathbf{P}$ and $\mathbf{V}$ run sumcheck to reduce checking the claim

$$\sum_{\boldsymbol{x}\in\{0,1\}^{\log n_{\mathsf{ERA}}}} \left(\hat{d}(\boldsymbol{x}) \cdot \left(\beta^2 + \beta \cdot \hat{\mathsf{id}}(\boldsymbol{x}) + \hat{w}^{\mathsf{perm}_2}(\boldsymbol{x})\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{x}, \boldsymbol{r}^{\mathsf{perm}_2}) = 0$$

to checking the following claim at a random point $\boldsymbol{r}^{\boldsymbol{d}}$ sampled by $\mathbf{V}$:

$$\left(\hat{d}(\boldsymbol{r}^{\boldsymbol{d}}) \cdot \left(\beta^2 + \beta \cdot \hat{\mathsf{id}}(\boldsymbol{r}^{\boldsymbol{d}}) + \hat{w}^{\mathsf{perm}_2}(\boldsymbol{r}^{\boldsymbol{d}})\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{r}^{\boldsymbol{d}}, \boldsymbol{r}^{\mathsf{perm}_2}) = \sigma^{\boldsymbol{d}}_{\mathsf{eval}} \ .$$

– $\mathbf{P}$ sends $\sigma^{\mathsf{perm}_2}_{\boldsymbol{d}} := \hat{d}(\boldsymbol{r}^{\boldsymbol{d}})$ and $\sigma^{\mathsf{perm}_2}_{\mathsf{perm}_2} := \hat{w}^{\mathsf{perm}_2}(\boldsymbol{r}^{\boldsymbol{d}})$ to $\mathbf{V}$, who checks that

$$\left(\sigma^{\mathsf{perm}_2}_{\boldsymbol{d}} \cdot \left(\beta^2 + \beta \cdot \hat{\mathsf{id}}(\boldsymbol{r}^{\boldsymbol{d}}) + \sigma^{\mathsf{perm}_2}_{\mathsf{perm}_2}\right) - 1\right) \cdot \mathrm{eq}(\boldsymbol{r}^{\boldsymbol{d}}, \boldsymbol{r}^{\mathsf{perm}_2}) = \sigma^{\boldsymbol{d}}_{\mathsf{eval}} \ .$$

- $\mathbf{P}$ and $\mathbf{V}$ set up inner product claims corresponding to the zerocheck evaluation claims:
  – $\mathsf{SplitClaimIP}(\boldsymbol{c}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{c}}), \sigma^{\mathsf{perm}_2}_{\boldsymbol{c}}, (\llbracket \boldsymbol{w}^{\boldsymbol{c}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$,
  – $\mathsf{SplitClaimIP}(\boldsymbol{\pi}_2, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{c}}), \sigma^{\mathsf{perm}_2}_{\boldsymbol{\pi}_2}, (\llbracket \boldsymbol{w}^{\boldsymbol{\pi}_2}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$,
  – $\mathsf{SplitClaimIP}(\boldsymbol{d}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{d}}), \sigma^{\mathsf{perm}_2}_{\boldsymbol{d}}, (\llbracket \boldsymbol{w}^{\boldsymbol{d}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$, and
  – $\mathsf{SplitClaimIP}(\boldsymbol{w}^{\mathsf{perm}_2}, \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{d}}), \sigma^{\mathsf{perm}_2}_{\mathsf{perm}_2}, (\llbracket \mathbf{o}^{\mathsf{perm}_2}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.
- $\mathbf{P}$ and $\mathbf{V}$ set up inner product claims that correspond to a zerocheck that enforces that $\boldsymbol{c}[j] = \boldsymbol{d}[j]$, for all $j \in [n_{\mathsf{ERA}}]$ by doing the following:
  – $\mathbf{P}$ sends $\sigma^{\mathsf{perm}_2}_{\boldsymbol{c},\boldsymbol{d}} := \hat{c}(\boldsymbol{r}^{\mathsf{perm}_2})$ to $\mathbf{V}$.
  – $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{c}, \mathbf{eq}(\boldsymbol{r}^{\mathsf{perm}_2}), \sigma^{\mathsf{perm}_2}_{\boldsymbol{c},\boldsymbol{d}}, (\llbracket \boldsymbol{w}^{\boldsymbol{c}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.
  – $\mathbf{P}$ and $\mathbf{V}$ run $\mathsf{SplitClaimIP}(\boldsymbol{d}, \mathbf{eq}(\boldsymbol{r}^{\mathsf{perm}_2}), \sigma^{\mathsf{perm}_2}_{\boldsymbol{c},\boldsymbol{d}}, (\llbracket \boldsymbol{w}^{\boldsymbol{d}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]})$.

Emulating $\hat{w}^{\mathsf{acc}_1}(\boldsymbol{X})$.

19. In the foregoing check, $\mathbf{P}$ is still left to convince $\mathbf{V}$ that $\hat{w}^{\mathsf{acc}_1}(\boldsymbol{r}^{\boldsymbol{c}}) = \sigma^{\mathsf{perm}_2}_{\mathsf{acc}_1}$. To do so, as explained in Section 6.2, $\mathbf{P}$ and $\mathbf{V}$ run

$$\mathsf{SplitClaimTIP}(\boldsymbol{w}^{\mathsf{perm}_1}, \boldsymbol{v}, \boldsymbol{A}_{\mathbf{eq}(\boldsymbol{r}^{\boldsymbol{c}})}, \sigma^{\mathsf{perm}_2}_{\mathsf{acc}_1}, (\llbracket \mathbf{o}^{\mathsf{perm}_1}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]}, (\llbracket \boldsymbol{w}^{\boldsymbol{v}}_i \rrbracket)_{i\in[\ell_{\mathsf{ERA}}]}) \ ,$$

where $\boldsymbol{A}_{\mathbf{eq}(\boldsymbol{r}^{\boldsymbol{c}})} := \sum_{i=1}^{n_{\mathsf{ERA}}} \mathbf{eq}(\boldsymbol{r}^{\boldsymbol{c}})[i] \cdot \boldsymbol{A}[i, \cdot]$ is the vector obtained by taking the linear combination of the rows of $\boldsymbol{A}$ with respect to the coefficients in $\mathbf{eq}(\boldsymbol{r}^{\boldsymbol{c}})$.

Checking the codeswitch claims.

20. $\mathbf{V}$ samples $\gamma \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathbf{P}$.
21. Define the 'codeswitch vector' $\boldsymbol{v}^{\mathsf{cs}} \in \mathbb{F}^{n_{\mathsf{ERA}}}$ to have the entry $\gamma^{\alpha_j - 1}$ at index α_j, for each $j \in [\nu]$, and 0 everywhere else.
22. $\mathbf{P}$ and $\mathbf{V}$ compute $\sigma^{\mathsf{cs}} := \sum_{j\in[\nu]} \gamma^{\alpha_j - 1} \cdot \sigma^{\mathsf{cs}}_j$.
23. $\mathbf{P}$ and $\mathbf{V}$ run sumcheck to reduce checking the claim

$$\sum_{\boldsymbol{b}\in\{0,1\}^{\log n_{\mathsf{ERA}}}} \hat{w}^{\mathsf{acc}_2}(\boldsymbol{b}) \cdot \hat{v}^{\mathsf{cs}}(\boldsymbol{b}) = \sigma^{\mathsf{cs}}$$

to checking the following claim at a random point $\boldsymbol{r}^{\mathsf{cs}}$ sampled by $\mathbf{V}$:

$$\hat{w}^{\mathsf{acc}_2}(\boldsymbol{r}^{\mathsf{cs}}) \cdot \hat{v}^{\mathsf{cs}}(\boldsymbol{r}^{\mathsf{cs}}) = \sigma^{\mathsf{cs}}_{\mathsf{eval}} \ .$$

24. $\mathbf{P}$ sends $\sigma^{\mathsf{cs}}_{\mathsf{acc}_2} := \hat{w}^{\mathsf{acc}_2}(\boldsymbol{r}^{\mathsf{cs}})$ to $\mathbf{V}$, who checks that

$$\sigma^{\mathsf{cs}}_{\mathsf{acc}_2} \cdot \hat{v}^{\mathsf{cs}}(\boldsymbol{r}^{\mathsf{cs}}) = \sigma^{\mathsf{cs}}_{\mathsf{eval}} \ .$$

Emulating $\hat{w}^{\mathsf{acc}_2}(\boldsymbol{X})$.

25. In the foregoing check, $\mathbf{P}$ is still left to convince $\mathbf{V}$ that $\hat{w}^{\mathsf{acc}_2}(\boldsymbol{r}^{\mathsf{cs}}) = \sigma^{\mathsf{cs}}_{\mathsf{acc}_2}$. To do so, $\mathbf{P}$ and $\mathbf{V}$ run

$$\mathsf{SplitClaimTIP}(\boldsymbol{w}^{\mathsf{perm}_2}, \boldsymbol{v}, \boldsymbol{A}_{\mathbf{eq}(\boldsymbol{r}^{\mathsf{cs}})}, \sigma^{\mathsf{cs}}_{\mathsf{acc}_2}, ([\![\mathbf{o}_i^{\mathsf{perm}_2}]\!])_{i\in[\ell_{\mathsf{ERA}}]}, ([\![\boldsymbol{w}_i^{\boldsymbol{v}}]\!])_{i\in[\ell_{\mathsf{ERA}}]}) \ ,$$

where $\boldsymbol{A}_{\mathbf{eq}(\boldsymbol{r}^{\mathsf{cs}})} := \sum_{i=1}^{n_{\mathsf{ERA}}} \mathbf{eq}(\boldsymbol{r}^{\mathsf{cs}})[i] \cdot \boldsymbol{A}[i,\cdot]$.

Figure 6: The subprotocol CodeswitchClaims.

PIOP soundness analysis. Having presented the details of CodeswitchOracles and CodeswitchClaims, we are now ready to analyze the soundness of the PIOP corresponding to the Reduce IOR. It is easy to see that the construction of CodeswitchClaims results from compiling such a PIOP using an interactive polynomial commitment based on the interleaved ERA code (using the terminology of the compiler in [BFRW25]). The soundness of the PIOP is analyzed as follows:

1. In steps 4-9, if

$$\sum_{\boldsymbol{y}=(\boldsymbol{y}_1,\boldsymbol{y}_2)\in\{0,1\}^{\log k}} \hat{g}(\boldsymbol{x}_1,\boldsymbol{y}_1)\cdot\hat{g}(\boldsymbol{x}_2,\boldsymbol{y}_2)\cdot\hat{m}(\boldsymbol{y}) \neq \hat{w}^{\mathcal{B}}(\boldsymbol{x}) \ ,$$

   then, unless with probability at most $\frac{\log n_{\mathcal{B}}}{|\mathbb{F}|}$, it must be that

$$\sum_{\boldsymbol{y}=(\boldsymbol{y}_1,\boldsymbol{y}_2)\in\{0,1\}^{\log k}} \hat{g}(\boldsymbol{r}_1^{\boldsymbol{x}},\boldsymbol{y}_1)\cdot\hat{g}(\boldsymbol{r}_2^{\boldsymbol{x}},\boldsymbol{y}_2)\cdot\hat{m}(\boldsymbol{y}) \neq \hat{w}^{\mathcal{B}}(\boldsymbol{r}^{\boldsymbol{x}}) \ .$$

   There are two cases to consider now:

   - $\hat{w}^{\mathcal{B}}(\boldsymbol{r}^{\boldsymbol{x}}) = \sigma^{\mathcal{B}}_{\mathsf{sc}}$. In this case, by the soundness guarantee of the sumcheck protocol, unless with probability at most $\frac{3\cdot\log k}{|\mathbb{F}|}$, it must be that

$$\hat{g}(\boldsymbol{r}_1^{\boldsymbol{x}},\boldsymbol{r}_1^{\boldsymbol{y}})\cdot\hat{g}(\boldsymbol{r}_2^{\boldsymbol{x}},\boldsymbol{r}_2^{\boldsymbol{y}})\cdot\hat{m}(\boldsymbol{r}^{\boldsymbol{y}}) \neq \sigma^{\mathcal{B}}_{\mathsf{eval}} \ .$$

     If $\sigma^{\mathcal{B}}_{\boldsymbol{g},1} = \hat{g}(\boldsymbol{r}_1^{\boldsymbol{x}},\boldsymbol{r}_1^{\boldsymbol{y}})$, $\sigma^{\mathcal{B}}_{\boldsymbol{g},2} = \hat{g}(\boldsymbol{r}_2^{\boldsymbol{x}},\boldsymbol{r}_2^{\boldsymbol{y}})$ and $\sigma^{\mathcal{B}}_{\boldsymbol{m}} = \hat{m}(\boldsymbol{r}^{\boldsymbol{y}})$, then $\mathbf{V}$ will reject, otherwise one of these equalities does not hold, and will be checked via a query to the respective oracles.
   - $\hat{w}^{\mathcal{B}}(\boldsymbol{r}^{\boldsymbol{x}}) \neq \sigma^{\mathcal{B}}_{\mathsf{sc}}$. In this case, this invalid claim is directly checked via a query to $\hat{w}^{\mathcal{B}}(\boldsymbol{r}^{\boldsymbol{x}})$.

2. Soundness of the permutation check in Step 14 follows directly by the soundness guarantee of the logUp permutation check. At the end of it, either the verifier rejects or one of the following claims are invalid: $\sigma^{\mathsf{perm}_1}_{\boldsymbol{a}} = \hat{a}(\boldsymbol{r}^{\boldsymbol{a}})$, $\sigma^{\mathsf{perm}_1}_{\boldsymbol{\pi}_1} = \hat{\pi}_1(\boldsymbol{r}^{\boldsymbol{a}})$, $\sigma^{\mathsf{perm}_1}_{\mathsf{rep}} = \hat{w}^{\mathsf{rep}}(\boldsymbol{r}^{\boldsymbol{a}})$, $\sigma^{\mathsf{perm}_2}_{\boldsymbol{c}} = \hat{c}(\boldsymbol{r}^{\boldsymbol{c}})$, $\sigma^{\mathsf{perm}_2}_{\boldsymbol{\pi}_2} = \hat{\pi}_2(\boldsymbol{r}^{\boldsymbol{c}})$, $\sigma^{\mathsf{perm}_2}_{\mathsf{acc}_1} = \hat{w}^{\mathsf{acc}_1}(\boldsymbol{r}^{\boldsymbol{c}})$, and $\sigma^{\mathsf{perm}_1}_{\boldsymbol{a},\boldsymbol{b}} = \hat{a}(\boldsymbol{r}^{\mathsf{perm}_1}) = \hat{b}(\boldsymbol{r}^{\mathsf{perm}_1})$. These claims are checked by querying the respective oracles.

3. The second permutation check in Step 18 follows by an entirely analogous argument, with the only difference being that, as explained in Section 6.2, the query to $\hat{w}^{\mathsf{acc}_1}$ is emulated via a sumcheck (and thus incurs a soundness error of $\frac{3}{|\mathbb{F}|}$ per round).

4. Finally, if $\hat{w}^{\mathsf{acc}_2}$ does not satisfy the codeswitch claims, the claim $\sum_{\boldsymbol{b}\in\{0,1\}^{\log n_{\mathsf{ERA}}}} \hat{w}^{\mathsf{acc}_2}(\boldsymbol{b}) \cdot \hat{v}^{\mathsf{cs}}(\boldsymbol{b}) = \sigma^{\mathsf{cs}}$ is invalid (unless with probability at most $\frac{n_{\mathsf{ERA}}}{|\mathbb{F}|}$), and then by the sumcheck protocol guarantee it must be that $\hat{w}^{\mathsf{acc}_2}(\boldsymbol{r}^{\mathsf{cs}}) \cdot \hat{v}^{\mathsf{cs}}(\boldsymbol{r}^{\mathsf{cs}}) \neq \sigma^{\mathsf{cs}}_{\mathsf{eval}}$. Then, if $\sigma^{\mathsf{cs}}_{\mathsf{acc}_2} = \hat{w}^{\mathsf{acc}_2}(\boldsymbol{r}^{\mathsf{cs}})$, the verifier rejects, while if not the final claim is invalid.

6.4 Optimizations

In the construction of the Reduce IOR presented in Fig. 4 (including the subprotocol CodeswitchClaims), we prioritized clarity of exposition over optimized efficiency. We now describe several optimizations that improve its concrete efficiency:

- **Using the logUp-GKR-based permutation check.** Since we use the logUp-based permutation check described in Section 6.1 to perform the two permutation checks in CodeswitchClaims, we must commit to four permutation witness vectors in total, namely $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{d}$. This contributes a sizeable overhead to the proof size, since committing to each of these vectors requires $\mathbf{P}$ to send ℓ_{ERA} oracles to $\mathbf{V}$, and $\mathbf{V}$ will have to query each of these oracles at several indices.

  In order to avoid this overhead, we can use a logUp-GKR-based permutation check. The logUp-GKR protocol was introduced in [PH23] as an improvement to logUp [Hab22]. In our context, the logUp-GKR-based permutation check can be summarized as follows: the permutation witness vectors in the logUp-based permutation check are required to run the fractional sumcheck protocol. Given multilinear polynomials $p(\boldsymbol{X}), q(\boldsymbol{X}) \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_{\log n_{\mathsf{ERA}}}]$, [PH23] show how to leverage (a bespoke version of) the GKR protocol to reduce checking that

$$\sum_{\boldsymbol{b}\in\mathbb{F}^{\log n_{\mathsf{ERA}}}} \frac{p(\boldsymbol{X})}{q(\boldsymbol{X})} = \sigma$$

  to checking one evaluation claim each about $p(\boldsymbol{X})$ and $q(\boldsymbol{X})$ at a random point, without sending any additional witness vectors. The cost of this GKR-based fractional sumcheck is having to sequentially run $\log n_{\mathsf{ERA}}$ many multivariate sumcheck protocols of the form

$$\sum_{\boldsymbol{b}\in\{0,1\}^i} f_i(\boldsymbol{b}) = \sigma_i \ ,$$

  for each $1 \leq i \leq \log n_{\mathsf{ERA}} - 1$. Here, for each $i \in [\log n_{\mathsf{ERA}} - 1]$, the polynomial $f_i(\boldsymbol{X})$ is an i-variate polynomial of degree at most 3 in each variable, that can be expressed as the sum of products of multilinear polynomials. It is well-known how to efficiently run the multivariate sumcheck protocol for such polynomials, and the proof size for the i-th multivariate sumcheck protocol with respect to $f_i(\boldsymbol{X})$ is $4 \cdot i$ field elements. Thus the total cost of using the GKR-based fractional sumcheck is $4 \cdot \left(\sum_{i=1}^{\log n_{\mathsf{ERA}}-1} i\right) = 2 \cdot (\log n_{\mathsf{ERA}} - 1) \cdot \log n_{\mathsf{ERA}}$ and one evaluation claim each about $p(\boldsymbol{X})$ and $q(\boldsymbol{X})$.

  In CodeswitchClaims, we would run such a GKR-based fractional sumcheck 4-times, where each time we set $p(\boldsymbol{X}) = 1$ and we set $q(\boldsymbol{X})$ to be $\alpha^2 + \alpha \cdot \hat{\pi}(\boldsymbol{X}) + \hat{a}(\boldsymbol{X})$, $\alpha^2 + \alpha \cdot \hat{\mathsf{id}}(\boldsymbol{X}) + \hat{b}(\boldsymbol{X})$, $\beta^2 + \beta \cdot \hat{\pi}(\boldsymbol{X}) + \hat{c}(\boldsymbol{X})$ and $\beta^2 + \beta \cdot \hat{\mathsf{id}}(\boldsymbol{X}) + \hat{d}(\boldsymbol{X})$ respectively. This would require us to run 4 copies of GKR-based fractional sumcheck instances, each with an i-th multivariate sumcheck claim of the form

$$\sum_{\boldsymbol{b}\in\{0,1\}^i} f_i^1(\boldsymbol{b}) = \sigma_i^1, \sum_{\boldsymbol{b}\in\{0,1\}^i} f_i^2(\boldsymbol{b}) = \sigma_i^2, \sum_{\boldsymbol{b}\in\{0,1\}^i} f_i^3(\boldsymbol{b}) = \sigma_i^3 \text{ and } \sum_{\boldsymbol{b}\in\{0,1\}^i} f_i^4(\boldsymbol{b}) = \sigma_i^4,$$

for each $i \in [\log n_{\mathsf{ERA}} - 1]$. However, we can batch the i-th multivariate sumcheck claims from the 4 GKR-based fractional sumcheck instances together, for each $i \in [\log n_{\mathsf{ERA}} - 1]$. This allows us to perform both permutation checks by just communicating $2 \cdot (\log n_{\mathsf{ERA}} - 1) \cdot \log n_{\mathsf{ERA}}$ field elements, and proving one evaluation claim about each of the four choices of $q(\boldsymbol{X})$.

- **Stacking oracles.** Another common IOPP optimization that can be implemented is to 'stack' all the oracles sent by $\mathbf{P}$ to $\mathbf{V}$. The motivation for this is as follows. A query to the output oracle $[\![\boldsymbol{w}']\!]$ of Reduce at index α is simulated by querying its component oracles

$$([\![\boldsymbol{w}_i]\!])_{i\in[\ell_{\boldsymbol{m}}]}, ([\![\boldsymbol{w}_i^{\mathsf{IP}}]\!])_{i\in[\mathsf{numIP}]}, ([\![\boldsymbol{w}_{i,1}^{\mathsf{TIP}}]\!])_{i\in[\mathsf{numTIP}]} \text{ and } ([\![\boldsymbol{w}_{i,2}^{\mathsf{TIP}}]\!])_{i\in[\mathsf{numTIP}]}$$

  at the same index α, and performing the appropriate linear combination of the obtained values. Therefore, instead of sending these oracles separately as vectors in $\mathbb{F}^{n_{\mathcal{C}'}}$, $\mathbf{P}$ can instead send a single 'stacked' oracle $[\![\boldsymbol{w}^{\mathsf{stacked}}]\!] \in (\mathbb{F}^{\ell_{\boldsymbol{m}}+\mathsf{numIP}+2\mathsf{numTIP}})^{n_{\mathcal{C}'}}$ at the end of the protocol. This can be interpreted as increasing the alphabet size of the IOR, so that in a single query, $\mathbf{V}$ can obtain the values of all the component oracles at the queried index.

  This is not just a syntactic optimization, but also translates to a significant improvement in the proof size of the multilinear polynomial commitment scheme (MLPCS) obtained by applying the BCS transform to our IOPP. This is because the Merkle commitment opening proofs for oracles are now batched together, whereas before, we would have to send separate opening proofs for each of the Merkle commitments to the component oracles.

  One subtlety is that the oracles output by the indexer naturally cannot be stacked together with the oracles sent by the prover during the interaction phase. As such, we have the indexer output a single stacked oracle $[\![\boldsymbol{w}^{\mathsf{stacked,ind}}]\!] \in (\mathbb{F}^{\ell_{\mathsf{ind}}})^{n_{\mathcal{C}'}}$ that consists of the indexer vectors

$$\left([\boldsymbol{w}_i^{\boldsymbol{g}}]_{i\in[\ell_{\boldsymbol{g}}]}, [\boldsymbol{w}_i^{\boldsymbol{\pi}_1}]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{w}_i^{\boldsymbol{\pi}_2}]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{w}_i^{\boldsymbol{v}}]_{i\in[\ell_{\mathsf{ERA}}]}\right) \ ,$$

  and $\mathbf{P}$ sends a single stacked oracle $[\![\boldsymbol{w}^{\mathsf{stacked},\mathbf{P}}]\!] \in (\mathbb{F}^{\ell_{\boldsymbol{m}}+\ell_{\mathsf{aux}}})^{n_{\mathcal{C}'}}$ that consists of the vectors

$$\left([\boldsymbol{w}_i]_{i\in[\ell_{\boldsymbol{m}}]}, [\boldsymbol{o}_i^{\mathcal{B}}]_{i\in[\ell_{\mathcal{B}}]}, [\boldsymbol{o}_i^{\mathsf{perm}_1}]_{i\in[\ell_{\mathsf{ERA}}]}, [\boldsymbol{o}_i^{\mathsf{perm}_2}]_{i\in[\ell_{\mathsf{ERA}}]}\right) \ .$$

  This also gives us the values of ℓ_{ind} and ℓ_{aux}, which turn out to be $\ell_{\mathsf{ind}} = \ell_{\boldsymbol{g}} + 3 \cdot \ell_{\mathsf{ERA}}$ and $\ell_{\mathsf{aux}} = \ell_{\boldsymbol{m}} + \ell_{\mathcal{B}} + 2 \cdot \ell_{\mathsf{ERA}}$.

6.5 Concrete efficiency of Reduce

To compute the concrete efficiency of the Reduce IOR, we need to analyze the precise costs of CodeswitchOracles and CodeswitchClaims, which we will do in this section.

- Firstly, recall that $\ell_{\boldsymbol{m}} = \lceil k/k' \rceil, \ell_{\boldsymbol{g}} = \left\lceil \sqrt{kn_{\mathcal{B}}}/k' \right\rceil, \ell_{\mathcal{B}} = \lceil n_{\mathcal{B}}/k' \rceil, \ell_{\mathsf{ERA}} = \lceil n_{\mathsf{ERA}}/k' \rceil$.
- Number of SplitClaimTIP invocations: 2 of size ℓ_{ERA}. Thus, $\mathsf{numTIP} = 2\ell_{\mathsf{ERA}}$.
- Number of SplitClaimIP invocations: 1 of size $\ell_{\boldsymbol{m}}$, 2 of size $\ell_{\boldsymbol{g}}$, 2 of size $\ell_{\mathcal{B}}$, and 7 of size ℓ_{ERA}. Thus, $\mathsf{numIP} = \ell_{\boldsymbol{m}} + 2\ell_{\boldsymbol{g}} + 2\ell_{\mathcal{B}} + 7\ell_{\mathsf{ERA}}$.

Output oracles. The prover sends two sets of oracles (all of purported codewords of $\mathcal{C}'$):

- The indexer oracles: there are $\ell_{\boldsymbol{g}} + 3\ell_{\mathsf{ERA}}$ many of these, which are stacked together.
- The online oracles: there are $\ell_{\boldsymbol{m}} + \ell_{\mathcal{B}} + 2\ell_{\mathsf{ERA}}$ many of these, which are stacked together.

Thus the virtual blowup (see Remark 5.3) of the output oracle is $t_{\mathsf{out}} := \ell_{\boldsymbol{m}} + \ell_{\mathcal{B}} + \ell_{\boldsymbol{g}} + 5\ell_{\mathsf{ERA}}$ in total.

Prover time. $\mathbf{P}$ encodes t_{out} many oracles under $\mathcal{C}'$, which costs $O(t_{\mathsf{out}} \cdot T_{\mathcal{C}'})$ time. Otherwise, $\mathbf{P}$ requires $O(\ell_{\mathsf{ERA}} \cdot n_{\mathsf{ERA}})$ time to do the remaining computation.

Query complexity. The verifier queries the input oracle $[\![\boldsymbol{w}]\!]$ at $\nu = \left\lceil \frac{-\lambda}{\log(1-\varepsilon)} \right\rceil$ indices in Fig. 4, and clearly no queries are made in CodeswitchOracles (Fig. 5) and CodeswitchClaims (Fig. 6). The virtual oracle blowup of $[\![\boldsymbol{w}]\!]$ is t_{in}, which results in a total query complexity of $\left\lceil \frac{-\lambda t_{\mathsf{in}}}{\log(1-\varepsilon)} \right\rceil$.

Non-oracle communication.

- The logUp-GKR-based permutation check costs $2 \cdot (\lceil \log n_{\mathsf{ERA}} \rceil - 1) \cdot \lceil \log n_{\mathsf{ERA}} \rceil$ field elements.
- The total communication cost from SplitClaimTIP invocations is numTIP field elements.
- The total communication cost from SplitClaimIP invocations is numIP field elements.
- There are 3 out-of-domain sample replies.
- The base code encoding check costs $3 \log k$ field elements for the sumcheck and 4 miscellaneous field elements.
- The codeswitch claim check costs $3\lceil \log n_{\mathsf{ERA}} \rceil$ field elements for the sumcheck and 1 miscellaneous field element.

Thus, the total communication cost of CodeswitchClaims is

$$\begin{aligned} &2 \cdot (\lceil \log n_{\mathsf{ERA}} \rceil - 1) \cdot \lceil \log n_{\mathsf{ERA}} \rceil + \mathsf{numIP} + \mathsf{numTIP} + 8 + 3 \log k + 3\lceil \log n_{\mathsf{ERA}} \rceil \\ =&(2\lceil \log n_{\mathsf{ERA}} \rceil + 1) \cdot \lceil \log n_{\mathsf{ERA}} \rceil + \mathsf{numIP} + \mathsf{numTIP} + 8 + 3 \log k \ . \end{aligned}$$

Adding the communication cost of Reduce outside of CodeswitchClaims, which is $4 \log k' + 3\ell_{\boldsymbol{m}} + \mathsf{numIP} + 2 \cdot \mathsf{numTIP}$ field elements (from Lemma 6.1), we get that the total communication cost of Reduce is

$$\begin{aligned} &(2\lceil \log n_{\mathsf{ERA}} \rceil + 1) \cdot \lceil \log n_{\mathsf{ERA}} \rceil + 2 \cdot \mathsf{numIP} + 3 \cdot \mathsf{numTIP} + 8 + 3 \log k + 4 \log k' + 3\ell_{\boldsymbol{m}} \\ =&(2\lceil \log n_{\mathsf{ERA}} \rceil + 1) \cdot \lceil \log n_{\mathsf{ERA}} \rceil + 2\ell_{\boldsymbol{m}} + 4\ell_{\boldsymbol{g}} + 4\ell_{\mathcal{B}} + 14\ell_{\mathsf{ERA}} + 6\ell_{\mathsf{ERA}} + 8 + 3 \log k + 4 \log k' + 3\ell_{\boldsymbol{m}} \\ =&(2\lceil \log n_{\mathsf{ERA}} \rceil + 1) \cdot \lceil \log n_{\mathsf{ERA}} \rceil + 5\ell_{\boldsymbol{m}} + 4\ell_{\boldsymbol{g}} + 4\ell_{\mathcal{B}} + 20\ell_{\mathsf{ERA}} + 8 + 3 \log k + 4 \log k' \ . \end{aligned}$$

Note that the overall proof size of the compiled MLPCS is usually dominated by the queries, and these extra communication costs are usually smaller.

7 Implementation and evaluation

Measuring concrete performance. To compare concrete performance with previous work, we compile IOPPs into *multilinear polynomial commitment schemes* (MLPCSs) via Merkle commitments. This gives a holistic view of the concrete efficiency of the IOPPs when deployed in practice. The proof size of the MLPCS is determined by (a) the query complexity of the IOPP, (b) the communication overhead of the IOPP, and (c) the overhead of the Merkle commitment scheme. Similarly, the prover time of the MLPCS is determined by (a) the prover time of the IOPP, (b) the encoding time of the code, and (c) the overhead of the Merkle commitment scheme. In this section, we use the phrase 'proof size of IOPP' to refer to the proof size of the resulting MLPCS obtained by compiling the IOPP via Merkle commitments.

In Section 7.1, we evaluate the performance of ERA codes by instantiating the Interleaved IOPP of Ligero [AHIV17] with it and comparing against the performance of other field-agnostic codes such as Brakedown's code [GLSTW23], Expand-Accumulate codes [BFKTWZ24], BaseFold's code [ZCF24]. We focus on four main metrics: (a) prover commitment time, (b) prover evaluation time, (c) total proof size, and (d) verifier time. Then, in Section 7.2, we estimate the concrete proof size of the MLPCS obtained from our IOPP for ERA codes and compare it with the proof sizes of BaseFold [ZCF24] and BrakingBase [NST25] when initialized with suitable parameters.

Our Rust implementation leverages the arkworks [arkworks] ecosystem. We use Blake3 for Merkle tree hashing and secp256k1 for field arithmetic. We used 8 Rayon threads for parallelization and the Criterion crate for benchmarking. All reported times are averaged over at least 10 runs. We benchmarked our implementation on a 2023 MacBook Air with an Apple M2 chip and 16GB of RAM, running macOS Sonoma 14.0. Our implementation is available at https://anonymous.4open.science/r/agnostir-F3B4/.

7.1 The Interleaved IOPP

We briefly recall the Interleaved IOPP of Ligero/Brakedown and explain how the choice of error-correcting code $\mathcal{C}$ affects the performance of the resulting IOPP. We phrase it as an *IOPP for random multilinear evaluation*, where the prover has a message $\boldsymbol{m}$ of length 2^k, and proves to the verifier that $\hat{m}(\boldsymbol{r}) = y$ for a random point $\boldsymbol{r}$, where $\hat{m}$ is the multilinear extension of $\boldsymbol{m}$. We obtain an interactive argument by instantiating the oracles with Merkle commitments; this can then be made non-interactive via Fiat–Shamir.

Let η be an interleaving parameter, and let the parameters of $\mathcal{C}$ be $(\mathbb{F}, 2^{n-\eta}, 2^{k-\eta}, \delta_{\mathcal{C}})$. The resulting interactive argument consists of two algorithms:

- **Commitment:** The prover divides $\boldsymbol{m}$ into 2^η blocks of length $2^{k-\eta}$ and encodes each block using $\mathcal{C}$, obtaining 2^η codewords of length $2^{n-\eta}$. We refer to this as an *interleaved encoding* of $\boldsymbol{m}$, and the resulting code can be thought of as a code over the extension field $\mathbb{F}^{2^\eta}$, with the same relative distance as $\mathcal{C}$. The prover then commits to the interleaved encoding via a Merkle commitment.
- **Evaluation:** Let $\boldsymbol{r} = (\boldsymbol{r}_1 \,\|\, \boldsymbol{r}_2)$ where $\boldsymbol{r}_1 \in \mathbb{F}^\eta$ and $\boldsymbol{r}_2 \in \mathbb{F}^{k-\eta}$.
  1. The prover "folds" the 2^η rows of the message using the multilinear Lagrange coefficients of $\boldsymbol{r}_1$, obtains $\boldsymbol{m}_{\boldsymbol{r}_1}$ of length $2^{k-\eta}$, and sends it to the verifier.
  2. The verifier computes $\boldsymbol{w}_{\boldsymbol{r}_1}$ as the encoding of $\boldsymbol{m}_{\boldsymbol{r}_1}$ under $\mathcal{C}$, samples ν column indices $\alpha_1, \ldots, \alpha_\nu \in [2^{n-\eta}]$ uniformly at random, and sends them to the prover.
  3. The prover replies with the corresponding ν columns of the interleaved encoding, together with a Merkle opening proof for each column.

4. The verifier accepts if the following checks pass, and rejects otherwise:
   - The Merkle opening proofs are valid.
   - For each $i \in [\nu]$, the value of the α_i-th column of the interleaved encoding is consistent with $\boldsymbol{w}_{\boldsymbol{r}_1}$.
   - $\hat{w}_{\boldsymbol{r}_1}(\boldsymbol{r}_2) = y$, where $\hat{w}_{\boldsymbol{r}_1}$ is the multilinear extension of $\boldsymbol{w}_{\boldsymbol{r}_1}$.

Parameter choices. We target $\lambda = 100$ bits of security (for both soundness and failure probability of the code's distance), and consider message length 2^{22}. We choose the interleaving parameter $\eta = 6$, which minimizes the proof size for the target soundness.

- **Brakedown:** We use Brakedown's code with distance 0.1 and inverse rate 2. We use Haböck's [Hab23] improved analysis of Brakedown's code to obtain the hyper-parameters for the code given the target distance, field size, and message length.
- **Expand-Accumulate:** We use the Expand-Accumulate code with distance 0.1 and inverse rate 2, which is the same code used in [BFKTWZ24]. We use the same hyper-parameters for the code as given in [BFKTWZ24].
- **BaseFold:** We use BaseFold's code with inverse rate 2, which results in a relative distance of 0.224. We use the analysis of BaseFold's code given in [ZCF24] to obtain these distance guarantees.
- **ERA:** We use Brakedown's code with distance 0.01 and inverse rate 1.1 as the base code $\mathcal{B}$. We use $r = 4$ as the repetition factor, which results in an ERA code with inverse rate 4.4 and relative distance of 0.226.

The distance analyses for these codes appear in the respective papers, and we additionally provide scripts for computing their concrete distance estimates here: https://anonymous.4open.science/r/era-code-analysis-1DFF.

Results. We use the results for mutual correlated agreement for arbitrary codes proven in [Zei24; GKL24]. Given a code with relative distance δ, these results imply that the number of columns queried by the verifier to achieve soundness $2^{-\lambda}$ is $q(\delta) := \left\lceil \frac{-\lambda}{\log_2(1-\delta)^{1/3}} \right\rceil = \left\lceil \frac{-3\lambda}{\log_2(1-\delta)} \right\rceil$.

Code	Commit Time	Prover Eval Time	Proof Size (KiB)	Verify Time	Verifier Queries
Brakedown	636.18 ms	119.78 ms	6431.1	48.587 ms	1974
EA	7.6476 s	116.13 ms	6431.1	525.00 ms	1974
BaseFold	675.61 ms	119.16 ms	4009.6	48.433 ms	869
ERA	733.67 ms	119.45 ms	3894.3	34.700 ms	804
Conjectured ERA	587.25 ms	119.79 ms	3894.3	23.162 ms	804

Table 3: Benchmarks for the interactive argument from the Interleaved IOPP for message length 2^{22} and interleaving parameter $\eta = 6$.

Remark 7.1 (Conjectured ERA)**.** We conjecture that the multiplication steps in the definition of ERA codes are not required to achieve the same (or even better) distance guarantees. The only place where we use the multiplication steps in our distance analysis is in the proofs of Lemmas 3.8 and 3.11: instead of showing that $> (1 - c/r)$-fraction of intervals are on-intervals with overwhelming probability over the choice of the multiplication vectors, we believe it is possible to show that $> (1 - 1/r - 1/\chi - o(1))$-fraction of intervals are on-intervals with overwhelming probability over the choice of the random permutations, where χ is the characteristic of the finite field (this is very large for secp256k1).

Remark 7.2 (BaseFold's performance)**.** Despite BaseFold's encoding time being $O(n \log n)$, its concrete encoding time is still competitive with the linear-time encodable codes we consider. This differs from the results reported in [ZCF24], where BaseFold's encoding time is about 3x that of Brakedown. We

implemented both the Brakedown and BaseFold encoders ourselves, so part of the discrepancy may come from implementation differences. A more likely explanation is that, over our choice of field, encoding time is dominated by the number of multiplications performed. For the instance sizes we consider, BaseFold and Brakedown perform a similar number of multiplications, which explains the similar encoding times we observe. We expect BaseFold to have worse performance for larger instance sizes, and also worse relative distance guarantees (unlike ERA codes whose distance guarantees improve with larger instance sizes).

7.2 Estimating the performance of our IOPP for ERA codes

In this section, we estimate the concrete proof size of our IOPP for ERA codes from Section 5 with $\ell = 1$ and compare it with the proof sizes of BaseFold [ZCF24] and BrakingBase [NST25] under suitable parameter choices.

Parameter choices. We target $\lambda = 100$ bits of security (for both soundness and failure probability of the code's distance). Let k be the message length, and η be the initial interleaving parameter. Define $\mathcal{C}$ as an ERA code with $r = 4$, message length k/η, and base code $\mathcal{B} = \mathcal{B}' \otimes \mathcal{B}'$, where $\mathcal{B}'$ is Brakedown's code with distance 0.01 and inverse rate 1.1.

For the evaluation proof, we perform one round of StartingIOR to reduce the initial $\mathsf{ME}^{(\mathcal{C}^\eta)}$ claim to an $\mathsf{ME}^{(\mathcal{C})}$ claim, and then one round of Reduce to reduce the $\mathsf{ME}^{(\mathcal{C})}$ claim to an $\mathsf{ME}^{(\mathsf{BF})}$ claim, where BF is BaseFold's code with certain parameters. Finally, we invoke BaseFold's IOPP to prove the $\mathsf{ME}^{(\mathsf{BF})}$ claim.

In addition to η, we need to choose the parameters of BF to define the protocol, which are its message length k' and its rate ρ_{BF}, and k_{final}, the terminating message length of BaseFold's IOPP (when the message is sent to the verifier in the clear). We therefore optimize over the choice of $\eta, k', \rho_{\mathsf{BF}}, k_{\mathsf{final}}$ to minimize the overall proof size. Note that the smaller the value of ρ_{BF}, the larger the distance of BF, and thus the smaller the number of queries required by the verifier in BaseFold's IOPP. However, we cannot choose ρ_{BF} arbitrarily small, since doing so would substantially increase the prover time in the evaluation phase of the IOPP. Accordingly, we restrict attention to choices of $\eta, k', \rho_{\mathsf{BF}}, k_{\mathsf{final}}$ for which the total oracle size of the evaluation proof is smaller than that of the initial interleaved encoding, ensuring that the overall prover time remains dominated by the initial commitment phase.

Proof size estimation example. As an example, we provide a detailed estimation of the proof size of our IOPP for ERA codes for a message length of $k = 2^{32}$. The optimal choice of parameters for this instance is given in Table 4.

Parameter	Optimized BaseFold	IOPP for ERA codes
$(\eta, k_{\mathsf{final}})$	$(2^5, 2^{12})$	$(2^6, 2^{12})$
ERA code parameters	N/A	$(\mathbb{F}, 2^{26} \cdot 4 \cdot 1.21, 2^{26}, 0.221817)$
BaseFold code parameters	$(\mathbb{F}, 2^{27} \cdot 2, 2^{27}, 0.152454)$	$(\mathbb{F}, 2^{25} \cdot 8, 2^{25}, 0.742531)$

Table 4: Chosen parameters for message length $k = 2^{32}$.

1. StartingIOR: From Lemma 5.5, we get that the proof of StartingIOR consists of $3 \log k = 96$ field elements.
2. Reduce: Looking at Fig. 4, there are three stacks of oracles that are queried by $\mathbf{V}$ in Reduce:
   (a) Output oracle of StartingIOR: this has a stack height of $\eta = 2^6$, and is queried at $q(0.2218) = 830$ indices. This results in $2^6 \cdot 830 = 53120$ field elements and 830 Merkle paths for a vector of length 2^{29} (the smallest power of 2 larger than the block length of ERA).

(b) Indexer oracles of Reduce: there are $\ell_{\boldsymbol{g}} + 3 \cdot \ell_{\mathsf{ERA}} = 33$ many of these, and queried at $q(0.7425) = 154$ many indices. This results in $33 \cdot 154 = 5082$ field elements and 154 Merkle paths for vectors of length 2^{28}.

(c) Online oracles of Reduce: there are $\ell_{\boldsymbol{m}} + \ell_{\mathcal{B}} + 2 \cdot \ell_{\mathsf{ERA}} = 25$ many of these, and queried at $q(0.7425) = 154$ many indices. This results in $25 \cdot 154 = 3850$ field elements and 154 Merkle paths for vectors of length 2^{28}.

In addition to the above, by Lemma 6.1, there are an additional $(2\lceil \log n_{\mathsf{ERA}} \rceil + 1) \cdot \lceil \log n_{\mathsf{ERA}} \rceil + 5\ell_{\boldsymbol{m}} + 4\ell_{\boldsymbol{g}} + 4\ell_{\mathcal{B}} + 20\ell_{\mathsf{ERA}} + 8 + 3 \log k + 4 \log k' = 2131$ field elements.

3. Proof size of BaseFold's IOPP: We terminate when the message length is $k_{\mathsf{final}} = 2^{12}$. Thus, we engage in 13 rounds of BaseFold's IOPP, and in each round, the verifier queries 154 many indices of an oracle (each index reveals 2 field elements), and the prover sends 3 field elements to describe the sumcheck round polynomial. This results in $2^{12} + 154 \cdot 13 \cdot 2 + 13 \cdot 3 = 4096 + 4004 + 39 = 8139$ field elements, and 154 Merkle paths for vectors of length $2^{28}, 2^{27}, \ldots, 2^{16}$ respectively in these 13 rounds.

Overall, the proof size is ≈ 4.014 MB, which consists of 72418 field elements and 53011 hashes.

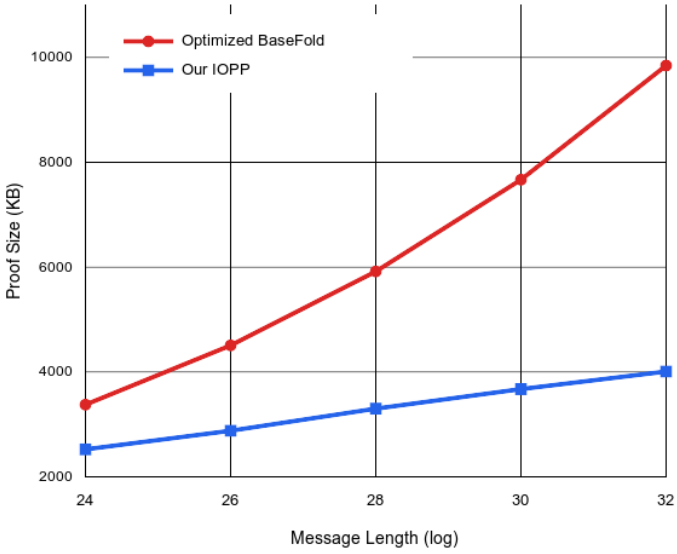


Figure 7: Estimated proof size of IOPP for ERA codes and optimized BaseFold for varying message lengths.

Comparison with BaseFold. For a fair comparison with BaseFold's IOPP, we optimize BaseFold's parameters for the same target soundness. We fix a BaseFold code of rate $1/2$, which results in a similar encoding time as the ERA code we use in the IOPP. The parameters we optimize for BaseFold's IOPP are the initial interleaving parameter η, and the termination message length k_{final}, which we choose to be 2^5 and 2^{12} respectively to minimize proof size. Thus, the initial message length of BaseFold's code for an instance size of 2^{32} is 2^{27}, which results in a relative distance of ≈ 0.1524 according to the analysis in [ZCF24]. The resulting proof size of this optimized BaseFold IOPP is ≈ 9.847 MB, which consists of 82233 field elements and 225472 hashes.[11] The two major reasons for this larger proof size compared to our IOPP for ERA codes

[11] This is significantly smaller than the proof sizes mentioned in [ZCF24] (for example, they report a proof size of ≈ 16 MB for an instance size of 2^{26}). This is mostly due to optimizing carefully over η and k_{final}.

are: (a) larger number of verifier queries required due to worse distance guarantees at the initial stage, and (b) codeswitching to BaseFold's code with smaller rate and better distance guarantees, which results in a much smaller proof size for our final IOPP compared to BaseFold's IOPP. In Fig. 7, we provide a comparison of the proof size of our IOPP for ERA codes and optimized BaseFold's IOPP for varying message lengths.

Comparison with BrakingBase. BrakingBase's IOPP [NST25] is similar to our IOPP except with two key differences: (a) it uses Brakedown's code as the initial code for StartingIOR, which has worse distance guarantees than ERA codes, and (b) it codeswitches to BaseFold's code with a large rate, thereby not fully leveraging the benefit of codeswitching. While BrakingBase reports significantly larger proof sizes than our IOPP (for example, ≈ 16 MB for a message length of 2^{28}), part of this gap comes from the second difference above, which could potentially be mitigated. However, the initial poor distance of Brakedown's code and the large proof size of their codeswitching protocol are inherent to their approach, and cannot be rectified without changing the underlying techniques.

Acknowledgements

We thank Matan Shtepel for his contributions to an earlier version of this work. We also thank Alex Block and Ulrich Haböck for providing us with the scripts for the distance analysis of EA codes and Brakedown's code respectively. Finally, we thank Hadas Zeilberger and Bhargav Thankey for providing us with concrete numbers for BaseFold's performance and BrakingBase's performance respectively.

AI use disclosure

We used OpenAI's Codex application with the 5.2 and 5.3 Codex models for coding and benchmarking. We used Microsoft Copilot in VSCode with a variety of models for auto-complete while writing the paper for grammatical and style suggestions, and for assisting with document-wide macro changes. Some proofs in Section A were proven with the assistance of Google Gemini 3 Pro. The authors take full responsibility for the content of the paper, and all the code and proofs were finalized and verified by the authors.

References

[ABF26] G. Arnon, D. Boneh, and G. Fenzi. *Open Problems in List-Decoding and Correlated Agreement*. Cryptology ePrint Archive, Report 2026/680. 2026. URL: https://eprint.iacr.org/2026/680.

[ACFY24] G. Arnon, A. Chiesa, G. Fenzi, and E. Yogev. "STIR: Reed–Solomon proximity testing with fewer queries". In: *Proceedings of the 44th Annual International Cryptology Conference*. CRYPTO '24. 2024, pp. 380–413. 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: *Proceedings of the 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '25. 2025, pp. 214–243. 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: *Proceedings of the 24th ACM Conference on Computer and Communications Security*. CCS '17. 2017, pp. 2087–2104. URL: https://eprint.iacr.org/2022/1608.

[AHKM25] K. Abbaszadeh, H. Hafezi, J. Katz, and S. Meiklejohn. "Single-Server Private Outsourcing of zk-SNARKs". In: *Proceedings of the 47th IEEE Symposium on Security and Privacy*. S&P '26. 2025. URL: https://eprint.iacr.org/2025/2113.

[AR11] A. G. i. Amat and E. Rosnes. “On the minimum distance properties of weighted nonbinary repeat multiple-accumulate codes”. In: *Information Theory and Applications Workshop, ITA 2011*. ITA ’11. 2011, pp. 135–139. URL: https://doi.org/10.1109/ITA.2011.5743575.

[arkworks] *arkworks zkSNARK ecosystem*. 2022. URL: https://arkworks.rs.

[AZ26] P. Akhiani and Y. Zhang. *Distance of RAA Codes over Large Finite Fields (with Applications in zkSNARKs and PCGs)*. Cryptology ePrint Archive, Paper 2026/524. 2026. URL: https://eprint.iacr.org/2026/524.

[BBHR18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. “Fast Reed–Solomon Interactive Oracle Proofs of Proximity”. In: *Proceedings of the 45th International Colloquium on Automata, Languages and Programming*. ICALP ’18. 2018, 14:1–14:17. URL: https://eccc.weizmann.ac.il/report/2017/134/.

[BCFRRZ25] M. Brehm, B. Chen, B. Fisch, N. Resch, R. D. Rothblum, and H. Zeilberger. “Blaze: Fast SNARKs from Interleaved RAA Codes”. In: *Proceedings of the 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT ’25. 2025, pp. 123–152. URL: https://eprint.iacr.org/2024/1609.

[BCFW25] B. Bünz, A. Chiesa, G. Fenzi, and W. Wang. “Linear-Time Accumulation Schemes”. In: *Proceedings of the 23rd Theory of Cryptography Conference*. TCC ’25. 2025. URL: https://eprint.iacr.org/2025/753.

[BCGGRS19] E. Ben-Sasson, A. Chiesa, L. Goldberg, T. Gur, M. Riabzev, and N. Spooner. “Linear-Size Constant-Query IOPs for Delegating Computation”. In: *Proceedings of the 17th Theory of Cryptography Conference*. TCC ’19. 2019. URL: https://eprint.iacr.org/2019/1230.

[BCKL22] E. Ben-Sasson, D. Carmon, S. Kopparty, and D. Levit. “Scalable and Transparent Proofs over All Large Fields, via Elliptic Curves (ECFFT Part II)”. In: *Proceedings of the 20th Theory of Cryptography Conference*. TCC ’22. 2022, pp. 467–496. URL: https://doi.org/10.1007/978-3-031-22318-1_17.

[BCKL23] E. Ben-Sasson, D. Carmon, S. Kopparty, and D. Levit. “Elliptic Curve Fast Fourier Transform (ECFFT) Part I: Low-degree Extension in Time O(n log n) over all Finite Fields”. In: *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’23. 2023, pp. 700–737. URL: https://doi.org/10.1137/1.9781611977554.ch30.

[BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner. “Interactive Oracle Proofs”. In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC ’16. 2016, pp. 31–60. URL: https://eprint.iacr.org/2016/116.

[BFKTWZ24] A. R. Block, Z. Fang, J. Katz, J. Thaler, H. Waldner, and Y. Zhang. “Field-Agnostic SNARKs from Expand-Accumulate Codes”. In: *Proceedings of the 44th Annual International Cryptology Conference*. CRYPTO ’24. 2024. URL: https://eprint.iacr.org/2024/1871.

[BFRW25] B. Bünz, G. Fenzi, R. D. Rothblum, and W. Wang. *TensorSwitch: Nearly Optimal Polynomial Commitments from Tensor Codes*. Cryptology ePrint Archive, Paper 2025/2065. 2025. URL: https://eprint.iacr.org/2025/2065.

[BGKS20] E. Ben-Sasson, L. Goldberg, S. Kopparty, and S. Saraf. “DEEP-FRI: Sampling Outside the Box Improves Soundness”. In: *Proceedings of the 11th Innovations in Theoretical Computer Science Conference*. ITCS ’20. 2020, 5:1–5:32. URL: https://eprint.iacr.org/2019/336.

[BMMS25] A. Baweja, P. Mishra, T. Mopuri, and M. Shtepel. *FICS and FACS: Fast IOPPs and Accumulation via Code-Switching*. Cryptology ePrint Archive, Paper 2025/737. 2025. URL: https://eprint.iacr.org/2025/737.

[BMNW25] B. Bünz, P. Mishra, W. Nguyen, and W. Wang. “Arc: Accumulation for Reed–Solomon Codes”. In: *Proceedings of the 45th Annual International Cryptology Conference*. CRYPTO ’25. 2025. URL: https://eprint.iacr.org/2024/1731.

[Boy+22] E. Boyle et al. “Correlated Pseudorandomness from Expand-Accumulate Codes”. In: *Proceedings of the 42nd Annual International Cryptology Conference*. CRYPTO ’22. 2022, pp. 603–633.

[DI14] E. Druk and Y. Ishai. “Linear-Time Encodable Codes Meeting the Gilbert-Varshamov Bound and Their Cryptographic Applications”. In: *Proceedings of the 5th Innovations in Theoretical Computer Science Conference*. ITCS ’14. 2014, pp. 169–182.

[DJM98] D. Divsalar, H. Jin, and R. J. McEliece. “Coding theorems for “turbo-like” codes”. In: *Proceedings of the annual Allerton Conference on Communication control and Computing*. Vol. 36. Allerton ’98. 1998, pp. 201–210.

[Eth] L. Ethereum. *Lean Consensus*. URL: https://leanroadmap.org/.

[FS24] M. Frigo and A. Shelat. *Anonymous credentials from ECDSA*. Cryptology ePrint Archive, Paper 2024/2010. 2024. URL: https://eprint.iacr.org/2024/2010.

[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. 2024. URL: https://eprint.iacr.org/2024/1810.

[GLSTW23] A. Golovnev, J. Lee, S. T. V. Setty, J. Thaler, and R. S. Wahby. “Brakedown: Linear-Time and Field-Agnostic SNARKs for R1CS”. In: *Proceedings of the 43rd Annual International Cryptology Conference*. CRYPTO ’23. 2023, pp. 193–226. URL: https://eprint.iacr.org/2021/1043.

[GNR26] K. Gurkan, A. Novakovic, and R. D. Rothblum. *Bolt: Faster SNARKs from Sketched Codes*. Cryptology ePrint Archive, Paper 2026/310. 2026. URL: https://eprint.iacr.org/2026/310.

[Hab22] U. Haböck. *Multivariate lookups based on logarithmic derivatives*. Cryptology ePrint Archive, Paper 2022/1530. 2022. URL: https://eprint.iacr.org/2022/1530.

[Hab23] U. Haböck. *Brakedown’s expander code*. Cryptology ePrint Archive, Paper 2023/769. 2023. URL: https://eprint.iacr.org/2023/769.

[HGQZ26] K. Huang, Y. Guo, W. Qu, and J. Zhang. *Bootstrapping-Free Blind PCS: Achieving Constant Depth and Linear Prover*. Cryptology ePrint Archive, Paper 2026/487. 2026. URL: https://eprint.iacr.org/2026/487.

[LFKN92] C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan. “Algebraic Methods for Interactive Proof Systems”. In: *Journal of the ACM* 39.4 (1992), pp. 859–868.

[NST25] V. Nair, A. Sharma, and B. Thankey. “BrakingBase - a linear prover, poly-logarithmic verifier, field agnostic polynomial commitment scheme”. In: *Proceedings of the 31st International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT ’25. 2025. URL: https://eprint.iacr.org/2024/1825.

[PH23] S. Papini and U. Haböck. *Improving logarithmic derivative lookups using GKR*. Cryptology ePrint Archive, Paper 2023/1284. 2023. URL: https://eprint.iacr.org/2023/1284.

[Plo60] M. Plotkin. “Binary Codes with Specified Minimum Distance”. In: *IRE Transactions on Information Theory* 6.4 (1960), pp. 445–450.

[QGZ26] W. Qu, Y. Guo, and J. Zhang. *Lightning, Field-Agnostic Super-Efficient Polynomial Commitment Scheme*. Cryptology ePrint Archive, Paper 2026/258. 2026. URL: https://eprint.iacr.org/2026/258.

[RR24] N. Ron-Zewi and R. Rothblum. “Local Proofs Approaching the Witness Length”. In: *Journal of the ACM* 71.3 (2024), p. 18. URL: https://eprint.iacr.org/2019/1062.

[RR25] N. Ron-Zewi and R. Rothblum. “Proving as fast as computing: succinct arguments with constant prover overhead”. In: *Journal of the ACM* 72.2 (2025), pp. 1–54. URL: https://eprint.iacr.org/2021/1673.

[RRR16] O. Reingold, R. Rothblum, and G. Rothblum. “Constant-Round Interactive Proofs for Delegating Computation”. In: *Proceedings of the 48th ACM Symposium on the Theory of Computing*. STOC ’16. 2016, pp. 49–62.

[Spi96] D. A. Spielman. “Linear-time encodable and decodable error-correcting codes”. In: *IEEE Transactions on Information Theory* 42.6 (1996). Preliminary version appeared in STOC ’95., pp. 1723–1731.

[Tha13] J. Thaler. “Time-Optimal Interactive Proofs for Circuit Evaluation”. In: *Proceedings of the 33rd Annual International Cryptology Conference*. CRYPTO ’13. 2013, pp. 71–89.

[ZCF24] H. Zeilberger, B. Chen, and B. Fisch. “BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes”. In: *Proceedings of the 44th Annual International Cryptology Conference*. CRYPTO ’24. 2024, pp. 138–169. URL: https://eprint.iacr.org/2023/1705.

[Zei24] H. Zeilberger. *Khatam: Reducing the Communication Complexity of Code-Based SNARKs*. Cryptology ePrint Archive, Paper 2024/1843. 2024. URL: https://eprint.iacr.org/2024/1843.

A Proof of E_{perm} claims

We restate and prove the claims about the exponent E_{perm}: the exponent term in the probability bound of $\mathcal{F}(\delta)$, as defined in Lemma 3.16. Note that c and r are constants that are fixed throughout, and we define $A := (1 - c/r)$ and $B := c/r$, which are used throughout in these proofs.

A.1 Useful facts

Fact A.1 (Titu's lemma / Sedrakyan's inequality)**.** For any real numbers x_1, x_2 and positive real numbers y_1, y_2, we have

$$\frac{x_1^2}{y_1} + \frac{x_2^2}{y_2} \geq \frac{(x_1 + x_2)^2}{y_1 + y_2} \ .$$

Fact A.2 (Jensen's inequality)**.** Let f be a strictly convex function over an interval $I \subseteq \mathbb{R}$. For any $z \in (0, 1)$ and any $x_1, x_2 \in I$, we have

$$zf(x_1) + (1 - z)f(x_2) \geq f(zx_1 + (1 - z)x_2) \ .$$

Fact A.3 (Derivatives of common functions)**.** Let $H(x) := -x \log_2 x - (1 - x) \log_2(1 - x)$ be the binary entropy function. Then, we have

$$\begin{aligned} \frac{d}{dx} H(x) &= \log_2 \left(\frac{1 - x}{x} \right) \\ \frac{d}{dx} x \cdot H(a/x) &= -\log_2(1 - a/x) \\ \frac{d}{dx}(b - x) \cdot H(a/(b - x)) &= \log_2(1 - a/(b - x)) \ . \end{aligned}$$

A.2 Deferred claims

Claim 3.17. Let n, ℓ_1, m_2 be fixed integers such that $m_2 < \ell_1$. Define

$$f_{\mathsf{inner}}(\ell_2) := \ell_2 \cdot H\left(\frac{m_2}{\ell_2} \right) + (n - \ell_2) \cdot H\left(\frac{\ell_1 - m_2}{n - \ell_2} \right) \ ,$$

Then, f_{inner} is increasing over $\ell_2 \in \left(m_2, \frac{m_2}{\ell_1} \cdot n \right)$.

Proof. Using Fact A.3, we compute the first derivative of f_{inner} with respect to ℓ_2:

$$\begin{aligned} f'_{\mathsf{inner}}(\ell_2) &= -\log_2 \left(1 - \frac{m_2}{\ell_2} \right) + \log_2 \left(1 - \frac{\ell_1 - m_2}{n - \ell_2} \right) \\ &= \log_2 \left(\frac{\ell_2 \cdot (n - \ell_2 - \ell_1 + m_2)}{(n - \ell_2) \cdot (\ell_2 - m_2)} \right) \ . \end{aligned}$$

We have

$$\frac{\ell_2 \cdot (n - \ell_2 - \ell_1 + m_2)}{(n - \ell_2) \cdot (\ell_2 - m_2)} > 1$$

$$\iff \ell_2 n - \ell_2^2 - \ell_1\ell_2 + m_2\ell_2 > \ell_2 n - \ell_2^2 - m_2 n + m_2\ell_2$$
$$\iff \ell_1\ell_2 < m_2 n \ .$$

Thus, if $\ell_2 < \frac{m_2}{\ell_1} \cdot n$, then $\frac{\ell_2 \cdot (n-\ell_2-\ell_1+m_2)}{(n-\ell_2)\cdot(\ell_2-m_2)} > 1$, which implies that $f'_{\mathsf{inner}}(\ell_2) > 0$ as desired. □

Claim 3.18. Let $r \in \mathbb{N}$ be a repetition parameter and let c be a positive constant such that

$$c \leq \frac{r}{2} \cdot \left(1 - \frac{1}{\sqrt{5}}\right) \ . \tag{8}$$

Let $\delta \leq (1 - c/r)$, and n, ℓ_1 be fixed integers such that $\ell_1 < \delta n/(1 - c/r)$. Define

$$f_{\mathsf{middle}}(m_2) := m_2 \cdot H\left(\frac{\ell_1 - m_2}{m_2}\right) + (\delta n) \cdot H\left(\frac{m_2}{\delta n}\right) + (n - \delta n) \cdot H\left(\frac{\ell_1 - m_2}{n - \delta n}\right) \ .$$

Then, f_{middle} is decreasing over $m_2 \in (\ell_1(1 - c/r), \ell_1)$.

Proof. We have

$$f'_{\mathsf{middle}}(m_2) = \log_2\left(\frac{(\ell_1 - m_2)^2(\delta n - m_2)}{(2m_2 - \ell_1)^2(n - \delta n - \ell_1 + m_2)}\right) \ .$$

Since $f'_{\mathsf{middle}}(m_2)$ is decreasing in $m_2 \in (\ell_1/2, \ell_1)$ (numerator decreases and denominator increases as m_2 increases), f_{middle} is concave over this range. Since $c/r < 1/2$, f_{middle} is concave over the range $(\ell_1(1 - c/r), \ell_1)$.

Thus, in order to show that f_{middle} is decreasing for $m_2 > \ell_1(1 - c/r)$, it suffices to show that $f'_{\mathsf{middle}}(\ell_1(1 - c/r)) < 0$. Replacing ℓ_1 with $d_1 n$ and plugging in $m_2 = \ell_1(1 - c/r)$, we have

$$\begin{aligned} f'_{\mathsf{middle}}(\ell_1(1 - c/r)) &= \log_2\left(\frac{(\ell_1 - \ell_1(1 - c/r))^2(\delta n - \ell_1(1 - c/r))}{(2\ell_1(1 - c/r) - \ell_1)^2(n - \delta n - \ell_1 + \ell_1(1 - c/r))}\right) \\ &= \log_2\left(\frac{(\ell_1 c/r)^2\,(\delta n - \ell_1(1 - c/r))}{(\ell_1(1 - 2c/r))^2\,(n - \delta n - \ell_1 c/r)}\right) \\ &= \log_2\left(\frac{c^2\,(\delta - d_1(1 - c/r))}{(r - 2c)^2\,(1 - \delta - d_1 c/r)}\right) \end{aligned}$$

Let $u = \frac{c}{r}$ and $v = \frac{(r-2c)^2}{c^2}$. $f'_{\mathsf{middle}}(\ell_1(1 - c/r)) < 0$ if the following holds:

$$\begin{aligned} & \frac{(r - 2c)^2\,(1 - \delta - d_1 c/r)}{c^2\,(\delta - d_1(1 - c/r))} > 1 \\ \iff\ & v \cdot \frac{1 - \delta - d_1 u}{\delta - d_1(1 - u)} > 1 \\ \iff\ & v \cdot (1 - \delta - d_1 u) > \delta - d_1(1 - u) \qquad \left[d_1 < \frac{\delta}{1 - u} \implies \delta - d_1(1 - u) > 0\right] \\ \iff\ & v - \delta v - \delta > d_1\,(uv + u - 1) \ . \end{aligned}$$

If $uv + u - 1 \geq 0$, then the right-hand side is non-negative. Since $d_1 < \delta/(1 - u)$, a sufficient condition for the above to hold is

$$v - \delta v - \delta > \frac{\delta}{1 - u}\,(uv + u - 1) \ ,$$

Rearranging again, the above is equivalent to

$$\delta < \frac{(1-u)v}{uv+u-1+(1-u)v+(1-u)} = 1-u = 1-\frac{c}{r} \ ,$$

which holds by our assumption.

All that remains is to be shown is that $uv+u-1 \geq 0$. Substituting back for u and v, we have

$$uv+u-1 = \frac{(r-2c)^2}{rc} + \frac{c}{r} - 1 = \frac{(r-2c)^2+c^2-rc}{rc} = \frac{r^2-5rc+5c^2}{rc} \ .$$

Thus, $uv+u-1 \geq 0$ if and only if $r^2-5rc+5c^2 \geq 0$. This is a quadratic in c with roots at $c = \frac{r}{2}\cdot(1 \pm \frac{1}{\sqrt{5}})$, so $r^2-5rc+5c^2 \geq 0$ if $c \leq \frac{r}{2}\cdot(1-\frac{1}{\sqrt{5}})$, which is satisfied by our assumption in (8). □

Claim 3.20. Consider any fixed $\delta < A^2$ and $d_1 \in [\delta_{\mathcal{B}} A, \delta/A)$. Then,

$$\frac{\partial^2}{\partial d_0^2} E_{\mathsf{perm}}(d_0, d_1, \delta) < 0 \text{ for all } d_0 \in (0, d_1/A) \ .$$

That is, $E_{\mathsf{perm}}(d_0, d_1, \delta)$ is concave in d_0 in the specified range.

Proof. We have

$$\begin{aligned}\frac{\partial}{\partial d_0}\left(E_{\mathsf{perm}}(d_0,d_1,\delta)/n\right) = \left(\frac{1}{r}-1\right)\log_2\left(\frac{1}{d_0}-1\right) + A\cdot H\left(\frac{B}{A}\right) \\ + A\cdot\log_2\left(\frac{d_1}{d_0 A}-1\right) + B\cdot\log_2\left(\frac{1-d_1}{d_0 B}-1\right) \ .\end{aligned}$$

Taking the second derivative, we get

$$\begin{aligned}\frac{\partial^2}{\partial d_0^2}\left(E_{\mathsf{perm}}(d_0,d_1,\delta)/n\right) &= \tfrac{1}{\ln 2}\cdot\left(-\tfrac{(1/r-1)}{d_0(1-d_0)} - \tfrac{A\cdot d_1}{(d_1-d_0A)\cdot d_0} - \tfrac{B\cdot(1-d_1)}{(1-d_1-d_0B)\cdot d_0}\right) \\ &= \tfrac{1}{\ln 2\cdot d_0}\cdot\left(\tfrac{1-1/r}{1-d_0} - \left(\tfrac{A}{(1-d_0A/d_1)} + \tfrac{B}{1-d_0B/(1-d_1)}\right)\right) \qquad (17)\end{aligned}$$

Applying Jensen's inequality (Fact A.2) to the function $f(x) = 1/(1-x)$, which is strictly convex for $x \in (0,1)$ and $z = A$, $x_1 = d_0A/d_1$, and $x_2 = d_0B/(1-d_1)$, we have

$$\begin{aligned}(17) &\leq \frac{1}{\ln 2\cdot d_0}\cdot\left(\frac{1-1/r}{1-d_0} - \frac{1}{1-(d_0A^2/d_1 + d_0B^2/(1-d_1))}\right) \\ &= \frac{1}{\ln 2\cdot d_0}\cdot\left(\frac{1-1/r}{1-d_0} - \frac{1}{1-d_0\left(\frac{A^2}{d_1}+\frac{B^2}{1-d_1}\right)}\right) \\ &\leq \frac{1}{\ln 2\cdot d_0}\cdot\left(\frac{1-1/r}{1-d_0} - \frac{1}{1-d_0(A+B)^2}\right) = \frac{-1}{r\cdot\ln 2\cdot d_0(1-d_0)} < 0 \ ,\end{aligned}$$

where the second inequality follows from Titu's lemma (Fact A.1) applied to $x_1 = A$, $x_2 = B$, $y_1 = d_1$, and $y_2 = 1-d_1$. Note that Jensen's inequality could be applied because $x_1 = d_0A/d_1 < 1$ and $x_2 = d_0B/(1-d_1) < \frac{\delta/A}{1-\delta/A}\cdot\frac{B}{A} = \frac{\delta B}{A^2-\delta A} < 1$, where the last inequality holds since $\delta < A^2$. □

Claim 3.21. Consider any fixed $\delta < A^2$ and $d_1 \in [\delta_{\mathcal{B}} A, \delta/A)$. Then, there exists a unique value $d_0^*(d_1) \in (0, d_1/A)$ such that

$$\frac{\partial}{\partial d_0} E_{\mathsf{perm}}(d_0, d_1, \delta)\Big|_{d_0 = d_0^*(d_1)} = 0 \ .$$

Note that $d_0^*(d_1)$ is a function of d_1, but not of δ. Furthermore, $d_0^*(d_1)$ is increasing in d_1.

Proof. Define $F(d_0, d_1) := \frac{\partial}{\partial d_0}\left(E_{\mathsf{perm}}(d_0, d_1, \delta)/n\right)$. From Claim 3.20, we have

$$F(d_0, d_1) = \left(\frac{1}{r} - 1\right) \log_2\left(\frac{1}{d_0} - 1\right) + A \cdot H\left(\frac{B}{A}\right) + A \cdot \log_2\left(\frac{d_1}{d_0 A} - 1\right) + B \cdot \log_2\left(\frac{1 - d_1}{d_0 B} - 1\right) \ .$$

Taking the limit as $d_0 \to 0^+$, we have

$$\lim_{d_0 \to 0^+} F(d_0, d_1) = \lim_{d_0 \to 0^+} \left(1 - \frac{1}{r}\right) \log_2(d_0) + A \cdot H\left(\frac{B}{A}\right) - \lim_{d_0 \to 0^+} \log_2(d_0) - \lim_{d_0 \to 0^+} \log_2(d_0) = +\infty \ .$$

Taking the limit as $d_0 \to (d_1/A)^-$, we have

$$\begin{aligned}\lim_{d_0 \to (d_1/A)^-} F(d_0, d_1) = \left(\frac{1}{r} - 1\right) \log_2\left(\frac{A}{d_1} - 1\right) + A \cdot H\left(\frac{B}{A}\right) \\ + \lim_{d_0 \to (d_1/A)^-} \log_2\left(\frac{d_1}{d_0 A} - 1\right) + \log_2\left(\frac{A(1 - d_1)}{d_1 B} - 1\right) = -\infty \ .\end{aligned}$$

Therefore, by the intermediate value theorem, there exists a value $d_0^*(d_1) \in (0, d_1/A)$ such that $F(d_0, d_1) = 0$.

To prove the second part of the claim, we apply the implicit function theorem to the equation $F(d_0, d_1) = 0$,

$$\frac{d d_0^*(d_1)}{d d_1} = -\frac{\frac{\partial}{\partial d_1} F(d_0, d_1)}{\frac{\partial}{\partial d_0} F(d_0, d_1)} \ .$$

The denominator is negative by Claim 3.20. The numerator is equal to

$$\begin{aligned}\frac{\partial}{\partial d_1} F(d_0, d_1) &= \frac{\partial}{\partial d_1} \frac{\partial}{\partial d_0} (E_{\mathsf{perm}}(d_0, d_1, \delta)/n) \\ &= \frac{1}{\ln 2} \cdot \left(\frac{A}{d_1 - d_0 A} - \frac{B}{1 - d_1 - d_0 B}\right) \ .\end{aligned}$$

This is positive if and only if $\frac{d_1}{A} - d_0 < \frac{1 - d_1}{B} - d_0$, which is equivalent to $d_1 < A$, which holds since $d_1 < \delta/A < A$. Therefore, $\frac{d d_0^*(d_1)}{d d_1} > 0$, completing the proof. □

Claim 3.22. Consider any fixed $d_0 \in [\delta_{\mathcal{B}}, \delta/A^2)$ and $d_1 \in [d_0 A, \delta/A)$. Then, we have

$$\frac{\partial}{\partial \delta} E_{\mathsf{perm}}(d_0, d_1, \delta) > 0 \ .$$

That is, $E_{\mathsf{perm}}(d_0, d_1, \delta)$ is increasing in δ in the specified ranges of d_0, d_1, for any feasible value of δ.

Proof. For any fixed d_0, d_1 specified in the claim, define the function

$$h_{d_0,d_1}(\delta) := \delta \cdot H\left(\frac{d_1 A}{\delta}\right) + (1-\delta) \cdot H\left(\frac{d_1 B}{1-\delta}\right) \ .$$

Using Fact A.3, we can compute the derivative with respect to δ:

$$h'_{d_0,d_1}(\delta) = -\log_2\left(1 - \frac{d_1 A}{\delta}\right) + \log_2\left(1 - \frac{d_1 B}{1-\delta}\right) = \log_2\left(\frac{1-\delta-d_1 B}{\delta - d_1 A}\right) \ . \tag{18}$$

Since $E_{\mathsf{perm}}(d_0, d_1, \delta)/n = h_{d_0,d_1}(\delta) + e(d_0, d_1)$ for some function $e(d_0, d_1)$ that does not depend on δ, we have $\frac{\partial}{\partial \delta}\left(E_{\mathsf{perm}}(d_0, d_1, \delta)/n\right) = h'_{d_0,d_1}(\delta)$. Thus, $E_{\mathsf{perm}}(d_0, d_1, \delta)$ is increasing in $\delta \in (d_1 A, A)$ if $h'_{d_0,d_1}(\delta) > 0$, which occurs if and only if

$$\frac{1-\delta-d_1 B}{\delta - d_1 A} > 1 \quad \Longleftrightarrow \quad \delta < \frac{1 - 2d_1 B + d_1}{2}$$

Since $d_1 < \delta/A$ by assumption, the above inequality hold if the following holds:

$$\delta < \frac{1 - 2(\delta/A)B + (\delta/A)}{2} \quad \Longleftrightarrow \quad \delta < A \ ,$$

which is true by assumption. □

Claim 3.23. Define the following function

$$g(d_1, \delta) := \sup_{d_0 \in (0, d_1/A)} E_{\mathsf{perm}}(d_0, d_1, \delta)/n \ . \tag{11}$$

Then, if $\delta < A$, we have

$$\frac{\partial}{\partial \delta}\frac{\partial}{\partial d_1} g(d_1, \delta) > 0 \text{ for all } d_1 \in [\delta_{\mathcal{B}} A, \delta/A) \ .$$

That is, $\frac{\partial}{\partial d_1} g(d_1, \delta)$ is increasing in δ in the specified range of d_1, for any feasible value of δ.

Proof. There are no terms in $E_{\mathsf{perm}}(d_0, d_1, \delta)$ that depend on both δ and d_0. Thus, if $d_0^*(d_1)$ is the maximizing value of d_0 for a fixed d_1 as defined in Claim 3.21, then we have

$$\begin{aligned}
\frac{\partial}{\partial \delta} g(d_1, \delta) &= \frac{\partial}{\partial \delta}\left(E_{\mathsf{perm}}(d_0^*(d_1), d_1, \delta)/n\right) \\
&= -\log_2\left(1 - \frac{d_1 A}{\delta}\right) + \log_2\left(1 - \frac{d_1 B}{1-\delta}\right) \qquad \text{[By (18) from Claim 3.22]}
\end{aligned}$$

Taking the partial derivative of $\frac{\partial}{\partial \delta} g(d_1, \delta)$ with respect to d_1, we have

$$\begin{aligned}
\frac{\partial}{\partial d_1}\frac{\partial}{\partial \delta} g(d_1, \delta) &= \frac{\partial}{\partial d_1}\left(-\log_2\left(1 - \frac{d_1 A}{\delta}\right) + \log_2\left(1 - \frac{d_1 B}{1-\delta}\right)\right) \\
&= \frac{1}{\ln 2} \cdot \left(\frac{A}{\delta - d_1 A} - \frac{B}{1 - \delta - d_1 B}\right) \ .
\end{aligned}$$

This is positive if and only if $\frac{A}{\delta - d_1 A} > \frac{B}{1-\delta-d_1 B}$, which is equivalent to $\delta < A$, which holds by assumption. Finally, the order of differentiation can be interchanged since the mixed partial derivatives are continuous, completing the proof. □

Claim 3.24. If $r \geq 4$ and $c < r/2$, there exists some sufficiently small $\delta_0 > 0$ such that

$$\frac{\partial}{\partial d_1} g(d_1, \delta_0) < 0 \text{ for all } d_1 \in [\delta_{\mathcal{B}} A, \delta_0 / A) \ ,$$

where $g(d_1, \delta)$ is defined in (11).

Proof. Using Fact A.3, we compute the first derivative of $g(d_1, \delta) := E_{\mathsf{perm}}(d_0^*(d_1), d_1, \delta)/n$ with respect to d_1, where $d_0^*(d_1)$ is defined as in Claim 3.21. Using the multivariate chain rule, we have

$$\begin{aligned}
&\frac{\partial}{\partial d_1} E_{\mathsf{perm}}(d_0^*(d_1), d_1, \delta) \\
&= \frac{\partial}{\partial d_1} E_{\mathsf{perm}}(d_0, d_1, \delta)\Big|_{d_0 = d_0^*(d_1)} + \frac{\partial}{\partial d_0} E_{\mathsf{perm}}(d_0, d_1, \delta)\Big|_{d_0 = d_0^*(d_1)} \cdot \frac{\partial}{\partial d_1} d_0^*(d_1) \\
&= \frac{\partial}{\partial d_1} E_{\mathsf{perm}}(d_0, d_1, \delta)\Big|_{d_0 = d_0^*(d_1)} \ ,
\end{aligned}$$

where the last step follows from the fact that $\frac{\partial}{\partial d_0} E_{\mathsf{perm}}(d_0, d_1, \delta^*)\Big|_{d_0 = d_0^*(d_1)} = 0$ by definition of $d_0^*(d_1)$. Therefore, we have

$$\begin{aligned}
\frac{\partial}{\partial d_1} g(d_1, \delta) &= \frac{\partial}{\partial d_1} E_{\mathsf{perm}}(d_0, d_1, \delta)\Big|_{d_0 = d_0^*(d_1)} \Big/ n \\
&= -\log_2\left(1 - \frac{d_0^*(d_1) A}{d_1}\right) + \log_2\left(1 - \frac{d_0^*(d_1) B}{1 - d_1}\right) \\
&+ A \cdot \log_2\left(\frac{\delta}{d_1 A} - 1\right) + B \cdot \log_2\left(\frac{1 - \delta}{d_1 B} - 1\right) + A \cdot H\left(\frac{B}{A}\right) + \log_2\left(\frac{d_1}{1 - d_1}\right) \\
&\leq -\log_2(1 - A) + A \cdot H\left(\frac{B}{A}\right) \\
&+ A \cdot \log_2\left(\frac{\delta}{A} - d_1\right) + B \cdot \log_2\left(\frac{1 - \delta}{B} - d_1\right) - \log_2(1 - d_1) \ ,
\end{aligned}$$

where the inequality follows from $d_0^*(d_1) \leq d_1$ (which we will show later). Now, the expression above is non-increasing in d_1, as the derivative with respect to d_1 is

$$\frac{-A^2}{\delta - d_1 A} - \frac{B^2}{1 - \delta - d_1 B} + \frac{1}{1 - d_1} \leq -\frac{(A + B)^2}{\delta - d_1 A + 1 - \delta - d_1 B} + \frac{1}{1 - d_1} = 0 \ ,$$

where the inequality follows from Titu's lemma (Fact A.1). Therefore, the maximum value of the above expression is attained at the smallest possible value of d_1, which is $d_1 = \delta_{\mathcal{B}} A$. Substituting this value, we get

$$\begin{aligned}
\frac{\partial}{\partial d_1} g(d_1, \delta) &\leq -\log_2(1 - A) + A \cdot H\left(\frac{B}{A}\right) \\
&+ A \cdot \log_2\left(\frac{\delta}{A} - \delta_{\mathcal{B}} A\right) + B \cdot \log_2\left(\frac{1 - \delta}{B} - \delta_{\mathcal{B}} A\right) - \log_2(1 - \delta_{\mathcal{B}} A) \ .
\end{aligned}$$

Clearly, as δ approaches $\delta_{\mathcal{B}} A^2$ from above, the term $\log_2\left(\frac{\delta}{A} - \delta_{\mathcal{B}} A\right)$ approaches $-\infty$, and hence the entire expression approaches $-\infty$.

All that remains is to show that $d_0^*(d_1) \leq d_1$ for all feasible d_1. For the sake of contradiction, assume otherwise, and let d_1' be such that $d_0^*(d_1') > d_1'$. Recall from Claim 3.20 that $d_0^*(d_1')$ is the unique solution to the following equation:

$$\left(\frac{1}{r}-1\right)\log_2\left(\frac{1}{d_0}-1\right)+A\cdot H\left(\frac{B}{A}\right)+A\cdot\log_2\left(\frac{d_1'}{d_0A}-1\right)+B\cdot\log_2\left(\frac{1-d_1'}{d_0B}-1\right)=0\ .$$

In Claim 3.20, we showed that the left-hand side is a strictly decreasing function of d_0, and therefore if d_0 is set to $d_1' < d_0^*(d_1')$ then the left-hand side must be positive. Substituting $d_0 = d_1'$ into the left-hand side, we get

$$\begin{aligned}&\left(\frac{1}{r}-1\right)\log_2\left(\frac{1}{d_1'}-1\right)+A\cdot H\left(\frac{B}{A}\right)+A\cdot\log_2\left(\frac{B}{A}\right)+B\cdot\log_2\left(\frac{1-d_1'}{d_1'B}-1\right)\\ \leq&\left(\frac{1}{r}-1\right)\log_2\left(\frac{1}{A^2}-1\right)+A\cdot H\left(\frac{B}{A}\right)+A\cdot\log_2\left(\frac{B}{A}\right)+B\cdot\log_2\left(\frac{1-A^2}{A^2B}-1\right)\ ,\end{aligned}$$

where the inequality follows from the fact that the expression is increasing in d_1' (can be verified by taking the derivative with respect to d_1'), and since $d_1' < d_0^*(d_1') < A^2$. It can be verified that this expression is < 0 for any $r \geq 4$ and $c < r/2$, which gives us the desired contradiction. □