Linear-Time Probabilistic Proofs with Sublinear Verification for Algebraic Automata Over Every Field

Jonathan Bootle
jbt@zurich.ibm.com
IBM Research – Zurich

Alessandro Chiesa
alessandro.chiesa@epfl.ch
EPFL

Ziyi Guan
ziyi.guan@epfl.ch
EPFL

Siqi Liu
sliu18@berkeley.edu
UC Berkeley

September 24, 2022

Abstract

Interactive oracle proofs (IOPs) are a generalization of probabilistically checkable proofs that can be used to construct succinct arguments. Improvements in the efficiency of IOPs lead to improvements in the efficiency of succinct arguments. Key efficiency goals include achieving provers that run in linear time and verifiers that run in sublinear time, where the time complexity is with respect to the arithmetic complexity of proved computations over a finite field $\mathbb{F}$.

We consider the problem of constructing IOPs for any given finite field $\mathbb{F}$ with a linear-time prover and polylogarithmic query complexity. Several previous works have achieved these efficiency requirements with $O(1)$ soundness error for NP-complete languages. However, constrained by the soundness error of the sumcheck protocol underlying these constructions, the IOPs achieve linear prover time only for instances in fields of size $\Omega(\log n)$. Recent work (Ron-Zewi and Rothblum, STOC 2022) overcomes this problem, but with linear verification time.

We construct IOPs for the algebraic automata problem over any finite field $\mathbb{F}$ with a linear-time prover, polylogarithmic query complexity, and sublinear verification complexity. We additionally prove a similar result to Ron-Zewi and Rothblum for the NP-complete language R1CS using different techniques. The IOPs imply succinct arguments for (nondeterministic) arithmetic computations over any finite field with linear-time proving (given black-box access to a linear-time collision-resistant hash function).

Inspired by recent constructions of reverse-multiplication-friendly embeddings, our IOP constructions embed problem instances over small fields into larger fields and adapt previous IOP constructions to the new instances. The IOP provers are modelled as random access machines and use precomputation techniques to achieve linear prover time. In this way, we avoid having to replace the sumcheck protocol.

Keywords: interactive oracle proofs; succinct arguments; linear-time prover; sublinear verification

Contents

1 Introduction

A succinct argument is a protocol where a prover aims to convince a verifier that a statement is correct, by communicating a short and efficiently verifiable proof, rather than a witness to the correctness of the statement. Succinct arguments have found numerous applications, such as verifiable computation schemes and electronic voting systems. Motivated by these applications, researchers strive to improve the efficiency of succinct arguments. The main efficiency measures are communication complexity, and prover and verifier running time. Amazingly, prior work on succinct arguments [**Groth10b**; **Lipmaa13**; **GennaroGPR12**; **Groth16**] culminates in constructions whose communication complexity and verifier time are independent of the statement to be proved. However, the prover complexity of these constructions remains a major bottleneck in practice, motivating research into linear-time prover algorithms.

All known constructions of succinct arguments with linear-time provers are interactive protocols where the prover and the verifier interact over multiple rounds. Building on [**GoldwasserKR15**; **Thaler13**], [**XieZZPS19**] constructed succinct arguments for layered arithmetic circuit evaluation with linear prover time, later extended to all circuits in [**ZhangWZZ20**]. The drawback is that these protocols' verify the correctness of polynomial-time computations, rather than NP statements. Later constructions target NP languages, and use the *Interactive Oracle Proof (IOP)* model, first introduced in [**BenSassonCS16**] and [**ReingoldRR16**]. In the IOP model, the prover sends *oracles* to the verifier, who queries the oracles at various points instead of reading them in their entirety. Various IOP constructions such as [**AmesHIV17**; **BenSassonBHR17**] encode computations using polynomial encodings which require quasilinear time to compute, and whose multiplicative properties facilitate protocol design. However, linear-time constructions must employ linear-time encodings, for which no constructions with multiplicative properties are known.

The first construction of a linear-time IOP for general arithmetic circuits in [**BootleCGGHJ17**] had square-root communication complexity in the circuit size, and was subsequently improved to sublinear [**BootleCG20**] and finally polylogarithmic [**BootleCL22**], with comparable verification times, using techniques from [**RonZewiR19**; **BenSassonBHR17**]. However, as in [**RonZewiR19**], [**BootleCG20**; **BootleCL22**] rely on the the [**LundFKN92**] sumcheck procotol. Due to implicit use of polynomial encodings, the soundness error of the sumcheck protocol is $O(\frac{\log n}{|\mathbb{F}|})$, which limits [**BootleCG20**; **BootleCL22**] to circuits defined over fields of size $\Omega(\log n)$. Recently, [**RonZewiR22**] solved this problem with a new multi-sumcheck protocol using linear-time encodings, which leads to an IOP for circuit satisfiability with linear verification time and soundness error $O(1)$ over any finite field. Thus, the development of IOPs with linear-time provers so far is entirely based on eliminating polynomial arithmetic in favour of linear-time encodings, at first, in order to avoid computational overheads, and later, to achieve non-trivial soundness guarantees over constant-size fields.

On the other hand, [**CascudoG21**] successfully constructs sound IOPs for NP statements defined over $\mathbb{F}_2$ by reducing them to NP statements over *polynomial-size fields* using specialised encodings called reverse multiplication-friendly embeddings (RMFEs) [**CascudoCXY18**]. They then apply IOP protocols over large fields. Due to looseness in the reductions and overheads associated with the RMFEs, [**CascudoG21**] gives a superlinear-time prover algorithm. However, their approach is attractive, as it allows reuse of existing protocols and implementations. We revisit the embedding strategy of [**CascudoG21**] to see whether it can be used to construct linear-time prover algorithms.

In this work, we construct IOPs with linear-time prover for the language of $\mathrm{R1CS}$ automata, and the NP-complete language $\mathrm{R1CS}$ defined over $\mathbb{F}_2$, by combining techniques from [**CascudoG21**] and previous works on linear-time IOPs, along with some new techniques that reduce the prover's computational overhead. Perhaps surprisingly, our work shows that linear-time proofs over constant-size fields can leverage multiplicative encodings of super-constant size, avoiding the computational overhead associated with encoding the entire witness by using precomputation techniques. Furthermore, our IOP for $\mathrm{R1CS}$ automata achieves sublinear verification. Previously, this was only known for $\mathrm{R1CS}$ for over large fields, and used preprocessing techniques [**BootleCG20**; **BootleCL22**]. Note that preprocessing is essential for sublinear verification time for $\mathrm{R1CS}$, because merely reading the instance requires linear time for the verifier.

Though our result for $\mathrm{R1CS}$ is similar to that of [**RonZewiR22**], we use different computational models and techniques. Our provers and verifiers are random-access machines while theirs are circuits. More comparisons between the two approaches are given in Section 1.2.

1.1 Our results

We construct linear-time IOPs for computations over *every* field (including the boolean field). The first IOP supports *rank-1 constraint satisfiability* (R1CS), a standard generalization of arithmetic circuit-SAT, where the "circuit description" is encoded in coefficient matrices. This NP-complete problem is widely used in the IOP literature since it efficiently expresses arithmetic circuits and is convenient for protocol design.

Definition 1.1. *The* $\mathrm{R1CS}$ *problem over a finite field* $\mathbb{F}$ *is as follows: given matrices* $A, B, C \in \mathbb{F}^{N \times N}$ *with at most* $M = \Omega(N)$ *non-zero entries, and an instance vector* x *over* $\mathbb{F}$*, does there exist a witness* w *over* $\mathbb{F}$ *such that* $z := (x, w) \in \mathbb{F}^N$ *and* $Az \circ Bz = Cz$*? Here "*$\circ$*" denotes the entry-wise product.*

Merely checking the validity of a witness by directly checking the R1CS condition costs $O(M)$ operations over $\mathbb{F}$, so "linear time" for R1CS means computations that cost no more than $O(M)$ operations over $\mathbb{F}$ (or the equivalent in bit operations).

The first main result of this paper is an IOP for R1CS, over any given field, that achieves linear-time proving (thus also linear-size proofs) and polylogarithmic query complexity. The notion of linear time is achieved over a random-access machine. A comparison with prior linear-time IOPs is given in Figure 1.

Theorem 1.2 (main)**.** *For every finite field* $\mathbb{F}$*, there is a public-coin IOP (with point queries), for* $\mathrm{R1CS}$ *instances whose matrices have* $M = \Omega(N)$ *nonzero entries, that has the following efficiency:*

- *soundness error is* $O(1)$*;*
- *round complexity is* $O(\log N)$*;*
- *answer alphabet is* $\mathbb{F}$*;*
- *proof length is* $O(N)$*;*
- *query complexity is* $O(\log(N)^2/\log|\mathbb{F}|)$*;*
- *the prover is a RAM with word length* $\Theta(\log N)$ *and* $O(1)$ *registers that runs in* $O(M+N)$ *time and* $O(N)$ *space;*
- *the verifier is a RAM with word length* $\Theta(\log N)$ *and* $O(1)$ *registers that runs in* $O(M+N)$ *time and* $O(N)$ *space.*

The theorem directly implies the existence of linear-time succinct arguments for R1CS over any field, via a known implication that involves combining IOPs and linear-time collision resistant hash functions (used as a black box) [**BootleCGGHJ17**]. Such hash functions are known to exist, e.g., under certain assumptions about finding short codewords in linear codes [**ApplebaumHIKV17**]; moreover, these candidate hash functions are not known to be insecure against quantum adversaries, making the succinct argument plausibly post-quantum secure.

Corollary 1.3 (succinct argument)**.** *Using any linear-time collision-resistant hash function with security parameter* λ *as a black box, there is an interactive argument for* $\mathrm{R1CS}$ *over any finite field* $\mathbb{F}$ *where:*

- *soundness error is* $O(1)$*;*
- *round complexity is* $O(\log N)$*;*
- *communication complexity is* $\mathsf{poly}(\lambda, \log(N)^2/\log|\mathbb{F}|)$*;*
- *the prover is a RAM with word length* $\Theta(\log N)$ *and* $O(1)$ *registers that runs in* $O(\lambda \log(M+N)+M+N)$ *time and* $O(N)$ *space;*
- *the verifier is a RAM with word length* $\Theta(\log N)$ *and* $O(1)$ *registers that runs in* $O(\lambda \log(M+N)+M+N)$ *time and* $O(N)$ *space.*

The second IOP supports the algebraic automata language, which verifies the transition function for an automata using R1CS coefficient matrices. This problem is previously discussed in [**BenSassonCGGRS19**], where they construct an IOP with linear proof length and polylogarithmic verification time with respect to the computation time of the corresponding automata.

Definition 1.4. *The* **algebraic automata problem** *over a finite field* $\mathbb{F}$ *is as follows: given matrices* $A, B, C \in \mathbb{F}^{w\times 2w}$*, a vector* $x \in \mathbb{F}^w$*, and a computation time* T*, does there exist an execution trace* $z\colon [T+1] \to \mathbb{F}^w$ *such that* $z(1) = x$ *and* $A(z(t), z(t+1)) \circ B(z(t), z(t+1)) = C(z(t), z(t+1))$ *for every* $t \in [T]$*?*

Our second result is an IOP for the algebraic automata problem that achieves linear-time proving and sublinear verification over any field. In comparison, [**BenSassonCGGRS19**] can only be extended to constant-size fields with an additional logarithmic factor in the proof length. In the context of this problem, since the automata encoded by the matrices are executed T times, we define "linear time" for algebraic automata to be computations that cost no more than $O(T)$ operations in $\mathbb{F}$.

Theorem 1.5 (main)**.** *For every finite field* $\mathbb{F}$*, and for every positive constant* $\epsilon > 0$*, there is a public-coin IOP (with point queries), for the algebraic automata problem with instances with* $w = O(1)$*, that has the following efficiency:*
- *soundness error is* $O(1)$*;*
- *round complexity is* $O(\log T)$*;*
- *answer alphabet is* $\mathbb{F}$*;*
- *proof length is* $O(T)$*;*
- *query complexity is* $O(\log(T)^2/\log|\mathbb{F}|)$*;*
- *the prover is a RAM with word length* $\Theta(\log T)$ *and* $O(1)$ *registers that runs in* $O(T)$ *time and* $O(T)$ *space;*
- *the verifier is a RAM with word length* $\Theta(\log T)$ *and* $O(1)$ *registers that runs in* $O(T^\epsilon)$ *time and* $O(T^\epsilon)$ *space.*

Figure 2 compares their IOP with the one we have in Theorem 1.5. As with R1CS, Theorem 1.5 directly implies a succinct argument for R1CS automata.

Corollary 1.6 (succinct argument)**.** *For any constant* $\epsilon > 0$*, using any linear-time collision-resistant hash function with security parameter* λ *as a black box, there is an interactive argument for the algebraic automata problem over any finite field* $\mathbb{F}$ *where:*
- *soundness error is* $O(1)$*;*
- *round complexity is* $O(\log T)$*;*
- *communication complexity is* $\mathsf{poly}(\lambda, \log(T)^2/\log|\mathbb{F}|)$*;*
- *the prover is a RAM with word length* $\Theta(\log T)$ *and* $O(1)$ *registers that runs in* $O(\lambda\log(T) + T)$ *time and* $O(T)$ *space;*
- *the verifier is a RAM with word length* $\Theta(\log T)$ *and* $O(1)$ *registers that runs in* $O(\lambda\log(T) + T^\epsilon)$ *time and* $O(T^\epsilon)$ *space.*

IOP	execution model	circuit encoding cost	prover cost	verifier cost	query complexity	field size
[**BootleCGGHJ17**]	RAM	$O(N)$ $\mathbb{F}$-ops	$O(N)$ $\mathbb{F}$-ops	$O(\sqrt{N})$ $\mathbb{F}$-ops	$O(\sqrt{N})$	$\Omega(N)$
[**BootleCG20**]	RAM	$O(N)$ $\mathbb{F}$-ops	$O(N)$ $\mathbb{F}$-ops	$O(N^\epsilon)$ $\mathbb{F}$-ops	$O(N^\epsilon)$	$\Omega(N)$
[**BootleCL22**]	RAM	$O(N)$ $\mathbb{F}$-ops	$O(N)$ $\mathbb{F}$-ops	$\mathsf{polylog}(N)$ $\mathbb{F}$-ops	$O(\log N)$	$\Omega(N)$
[**XieZS22**]	RAM	$O(N)$ $\mathbb{F}$-ops	$O(N)$ $\mathbb{F}$-ops	$O(\log^2 N)$ $\mathbb{F}$-ops†	$O(\log^2 N)$	$\Omega(N)$
[**RonZewiR22**]	circuit	not applicable	$O(N)$ $\mathbb{F}$-ops	$O(N)$ $\mathbb{F}$-ops‡	$\mathsf{polylog}(N)$	any
Theorem 1.2	RAM	not applicable	$O(N)$ $\mathbb{F}$-ops	$O(N)$ $\mathbb{F}$-ops	$O(\log^2 N)$	any

Figure 1: Comparison of known IOPs with a linear-time prover. The parameters are for an N-gate arithmetic circuit defined over a field $\mathbb{F}$; and ϵ is any positive constant. The sublinear verification in all cases is achieved in the holographic setting (the verifier has oracle access to an encoding of the circuit). For the IOPs modelled using RAM, prover and verifier costs are measured in terms of the equivalent number of operations over $\mathbb{F}$, by dividing the total number of basic RAM operations by the number required to perform a field operation. (†: we compare against a preprocessing version of [**XieZS22**] incorporating techniques from [**Setty20**].) (‡: see Remark 1.8.)

IOP	execution model	prover cost	verifier cost	query complexity	field size
[**BenSassonCGGRS19**]	RAM	$O(T \log T)$ $\mathbb{F}$-ops	$\mathsf{polylog}(T)$ $\mathbb{F}$-ops	$O(1)$	$\Omega(T)$
[**HolmgrenR22**]	circuit	$O(T)$ $\mathbb{F}$-ops	$O(T^\epsilon)$ $\mathbb{F}$-ops	$O(T^\epsilon)$	any
Theorem 1.5	RAM	$O(T)$ $\mathbb{F}$-ops*	$O(T^\epsilon)$ $\mathbb{F}$-ops*	$O(\log^2 T)$	any

Figure 2: Comparison of known IOPs for R1CS automata. The parameters are for an automata with computation time T defined over a field $\mathbb{F}$; and ϵ is any positive constant. For the IOPs modelled using RAM, prover and verifier costs are measured in terms of the equivalent number of operations over $\mathbb{F}$, by dividing the total number of basic RAM operations by the number required to perform a field operation. (†: see Remark 1.8.)

1.2 Related works

[Page numbers for [HolmgrenR22] and [XieZS22] are missing in the bibliography as the proceedings does not seem to be online yet. —*Jonathan*] ★

Prior work on linear-time IOPs. Our main results focus on constructing efficient interactive oracle proofs with linear-time provers for R1CS. Several prior works have considered the same goal. First, [**BootleCGGHJ17**] obtained an IOP with linear-time prover and square-root verification time for arithmetic circuit satisfiability. Later, [**BootleCG20**] and [**BootleCL22**] improved this result by achieving sublinear, and then polylogarithmic verification time for R1CS. These works, [**BootleCGGHJ17**; **BootleCG20**; **BootleCL22**], all require non-constant-size fields. Recently, [**RonZewiR22**] proved a similar result to Theorem 1.2, and constructed an IOP with a linear-time prover for boolean circuit satisfiability, via different techniques. Figure 1 summarizes the properties of the IOPs obtained from these works.

We now carefully discuss how our result differs from the one proved by Ron-Zewi and Rothblum in [**RonZewiR22**]. Note that the comparison below is only between Theorem 1.2 and Theorem 1.7, since their work does not mention sublinear verification and targets only general but not staircase R1CS instances.

Theorem 1.7 ([**RonZewiR22**])**.** *There is an IOP for proving the satisfiability of a boolean circuit C where: soundness error is $O(1)$; round complexity is $O(\log |C|)$; proof length is $O(|C|)$ over the binary alphabet; query complexity is* $\mathsf{polylog}\,|C|$*; and the prover and verifier can be implemented via boolean circuits of size $|C|$ (that can be efficiently computed from C).*

Both Theorem 1.2 and Theorem 1.7 extend the line of works on linear-time IOPs by contributing new tools that enable a constant soundness error over boolean fields. The IOP in [**RonZewiR22**] works for the satisfiability of boolean circuits; however, it could be extended to work with $\mathrm{R1CS}$, the language our IOPs are constructed for. Meanwhile, we note that the execution models of the IOPs differ. The IOP prover in [**RonZewiR22**] is a boolean circuit that can be efficiently obtained from the boolean circuit whose satisfiability is being proved. Our IOP prover is an algorithm for a random-access machine. The [**RonZewiR22**] result would also imply a linear-time RAM prover if the linear-size prover circuit can be efficiently generated by a linear-time RAM program. We do not know whether our result implies a corresponding result with circuit prover.

More interestingly, the two results differ significantly in the underlying techniques.

- *Multi-sumcheck.* The IOP in [**RonZewiR22**] relies on a multi-sumcheck protocol based on code-switching [**RonZewiR19**], in combination with an arithmetization inspired by [**BootleCGGHJ17**]. The multi-sumcheck protocol foregoes the use of multiplication codes (the product of codewords is a codeword from a related code) of super-constant length, relying instead on linear-time encodable codes and repeated use of code-switching to avoid the computational overhead caused by multiplications of long codewords.

- *Embedding and precomputation.* Our IOP combines ideas from the tensor-to-point-query approach [**BootleCG20**], which itself is an optimization of the code-switching technique in [**RonZewiR19**], and the embedding techniques in [**CascudoG21**]. This leads to a construction over a field extension that is not too large, and we rely on precomputation to avoid costly multiplications in the extension field. Our IOP construction does use a generalization of the biased generator for boolean fields constructed in [**RonZewiR22**].

Remark 1.8. The verifier of the IOP in [**RonZewiR22**] and the verifier of our IOP for $\mathrm{R1CS}$ both run in linear time. This is optimal because the verifier must read the circuit/$\mathrm{R1CS}$ description. In both cases it remains an open question whether sublinear verification can additionally be achieved for general $\mathrm{R1CS}$ instances. Meanwhile, our IOP for algebraic automata achieve sublinear verification (without holography). We remark that [**RonZewiR22**] additionally discuss a model where the verifier performs a *private* linear-time computation in an offline phase followed by a sublinear-time computation in the online phase. This is a type of preprocessing that many probabilistic proofs satisfy (including our IOP) but we do not discuss this model since it is of limited use.

Concurrent work on linear-time IOPs. Recently, [**XieZS22**] developed new algorithms to improve the efficiency of the linear-time encodings used in linear-time IOPs, modified the proof composition techniques used in [**BootleCL22**], and produced an efficient implementation of the resulting arguments. Note that while [**XieZS22**] presents a protocol with linear verification time, it is straightforward to extend this to the results in Figure 1 using techniques from [**Setty20**], leading to the same dependence on large fields as in [**BootleCGGHJ17**; **BootleCG20**; **BootleCL22**].

Our work and [**RonZewiR22**] achieve constant soundness error, but can achieve $2^{-\lambda}$ soundness error with $O(\lambda) \cdot N$ prover time via repetition. Recently, [**HolmgrenR22**] improved the trade-off between the prover overhead and soundness error for special classes of R1CS instances including algebraic automata (referred to as succinct R1CS). They construct an IOP with $\mathsf{polylog}(\lambda) \cdot N$ prover time, $2^{-\lambda}$ soundness error, and sublinear verification time for R1CS instances whose matrices have a tensor structure. Setting $\lambda = O(1)$ leads to the results in Figure 2. Their general strategy is to improve the protocol in [**RonZewiR22**] by transforming input vectors for several subprotocols into tensor products. They then reduce soundness error by partitioning the tensors into $\Theta(\lambda)$ parts and applying the subprotocols to each part independently.

Whether there are IOPs for general R1CS instances achieving the same trade-off between prover time and soundness error is an open question.

Prior work on cryptographic proofs with linear-time provers. As previously discussed, constructions of linear-time IOPs can be compiled into linear-time succinct arguments using suitable collision resistant hash functions. One can also use more complex cryptographic compilers which involve further composition with succinct arguments. Prior works [**LeeSTW21**; **GolovnevLSTW21**] apply this strategy to the arguments from [**BootleCG20**]. Among other optimizations, they contribute improved constructions of linear-time encodable codes, which improve the concrete efficiency of linear-time IOPs and lead to efficient implementations. These works give arguments over large fields.

Another line of work focuses on achieving practical linear-size proofs with linear-time prover algorithms. For example, [**WengYKW20**; **BaumMRS21**; **YangSWW21**] provide proof systems for verifying arithmetic circuits over fields of any size, based on *vector oblivious linear evaluation* protocols. In particular, [**BaumMRS21**] also employ embedding techniques. These protocols have a linear-time verifier. Another strategy for achieving linear time prover and verifier complexity in earlier works comes from the MPC-in-the-head paradigm [**IshaiKOS09**]. Other works such as [**FranzeseKLO0W21**; **HeathK20a**] provide linear-time provers for statements about RAM executions.

2 Techniques

We describe the main ideas behind our result.

2.1 Starting point: the tensor-query to point-query paradigm

Our construction follows the same approach as [**BootleCG20**], first constructing a tensor IOP for R1CS with linear time and then compiling it to a standard IOP. This approach is an optimization of both the code-switching technique in [**RonZewiR19**], and the compiler for linear queries on vectors to point queries in [**BootleCGGHJ17**], tailored to this type of construction.

Tensor IOP for $\mathbf{R1CS}$. In a tensor IOP, the verifier makes tensor queries to oracle messages sent by the prover. A tensor IOP is parametrized by a field $\mathbb{F}_p$, where p is an arbitrary prime power, and positive integers k and t. Given a proof message $\Pi \in \mathbb{F}_p^{k^t}$, the verifier can make tensor queries of the form $q = (q_1, \ldots, q_t) \in (\mathbb{F}_p^k)^t$ and receive answer $\langle q_1 \otimes \cdots \otimes q_t, \Pi \rangle$. Recall that for standard IOPs, the verifier uses point queries which allow them to query single locations of proof messages. For R1CS, the problem decides whether for given coefficient matrices $A, B, C \in \mathbb{F}_p^{N \times N}$ and an instance vector $x \in \mathbb{F}_p^*$, there exists a witness vector w such that $z = (x, w) \in \mathbb{F}_p^N$ satisfies $Az \circ Bz = Cz$. Following the standard approach for designing IOPs for R1CS, it suffices for the prover $\mathbf{P}$ to send z and $z_A, z_B, z_C \in \mathbb{F}_p^N$ and convince the verifier $\mathbf{V}$ that $z_A = Az, z_B = Bz, z_C = Cz$, and $z_A \circ z_B = z_C$.

Twisted scalar product. All four conditions can be reduced to a relation called the twisted scalar product relation introduced by [**BootleCG20**], which checks whether $\vec{a}, \vec{b}, \vec{y} \in \mathbb{F}_p^n$ and $\tau \in \mathbb{F}_p$ satisfy $\langle \vec{a} \circ \vec{y}, \vec{b} \rangle = \tau$. The tensor IOP constructed by [**BootleCG20**] for this relation achieves a soundness error of $\log N / |\mathbb{F}_p|$.

- $z_U = Uz$: To check this type of condition, the verifier sends a random challenge r with tensor structure, reducing the conditions to $\langle r, z_U \rangle = \langle r^\intercal U, z \rangle$, where the left-hand side can be obtained through a tensor query to z_U. Then set $\vec{a} = r^\intercal U, \vec{y} = 1^n, \vec{b} = z$ and $\tau = \langle r, z_U \rangle$ and invoke the tensor IOP protocol to check that the right-hand side is equal to the left.
- $z_A \circ z_B = z_C$: This condition is checked by picking a random vector as the twist $\vec{y}$ and setting $\vec{a} = z_A, \vec{b} = z_B$, and $\tau = \langle z_C, \vec{y} \rangle$.

Tensor IOP to point IOP compiler. After obtaining a tensor IOP for R1CS, [**BootleCG20**] provides a way to efficiently convert the tensor IOP into a standard point IOP by simulating tensor queries via a number of point queries. More specifically, [**BootleCG20**] designs a compiler that takes in a tensor IOP and any linear code whose encoding function is represented by a circuit as inputs, and outputs a point IOP that decides the same language as the tensor IOP up to an overhead in soundness error.

Bottlenecks from [BootleCG20]. We cannot directly apply the linear-time IOP of [**BootleCG20**] to constant-size fields. This is because their final construction gives a tensor IOP with soundness error $\log N / |\mathbb{F}_p|$ due to the soundness error of the twisted scalar product protocol, which compiles to a standard IOP with soundness error $N / |\mathbb{F}_p|$. Therefore, the size of the underlying field has to be large in order for the soundness error to be small. Therefore, to adapt this approach for small fields, we will use an embedding from a small field to a larger field, inspired by [**CascudoG21**], and then design our IOPs in this larger field.

2.2 Overview of techniques

Recently, [**CascudoCXY18**] introduced embeddings called reverse-multiplication friendly embeddings (RMFEs) which were later used by [**CascudoG21**] to construct a Reed-Solomon-encoded IOP for R_{R1CS}

over $\mathbb{F}_2$. RMFEs are embeddings that transform a vector over a small field to an element in a large field while preserving multiplicative relations. More precisely, the encoding function $\phi\colon \mathbb{F}_2^a \to \mathbb{F}_{2^b}$ and decoding function $\psi\colon \mathbb{F}_{2^b} \to \mathbb{F}_2^a$ are such that $\vec{x} \circ \vec{y} = \psi(\phi(\vec{x})\phi(\vec{y}))$ for all $\vec{x}, \vec{y} \in \mathbb{F}_2^a$. The key idea behind the IOP construction is to embed the R_{R1CS} instance matrices and witness vectors into a large field $\mathbb{F}_{2^b}$ via the encoding function. Note that the embedding function ϕ preserves multiplicative relations between embedded vectors, which makes it possible for the IOP to check the relation $Az \circ Bz = Cz$.

Further, after the reduction, in addition to equations proving R_{R1CS} over $\mathbb{F}_{2^b}$, the prover needs to convince the verifier of some new relations. For instance, the prover needs to show that some given vector $\widetilde{x}$ is indeed a valid encoding in $\mathrm{Im}(\phi)$. To this end, [**CascudoCXY18**] introduces a subprotocol called a modular lincheck protocol for all such relations.

Unfortunately the IOP construction in [**CascudoCXY18**] uses Reed–Solomon codes, which immediately make the prover's running time superlinear. Additionally, two places in the modular lincheck protocol also incur superlinear operations already.

We get around using Reed–Solomon codes by applying the paradigm of [**BootleCG20**]. We first construct a tensor IOP and then use a tensor-to-point query compiler to obtain an IOP for R_{R1CS} over small fields.

The other two problems are caused by an inefficient use of randomized hashing in the modular lincheck protocol, and we solve them by first using a RMFE that has a *systematic* property and second by replacing the hash function with one that achieves smaller soundness error. The precise definition of a systematic RMFE is given in Section 2.3. More details on the construction of our modular lincheck protocol can be found in Section 2.4.

2.3 Systematic reverse-multiplication-friendly embeddings

In order to tackle the task for finite fields $\mathbb{F}_p$, we use reverse-multiplication-friendly embeddings (RMFE) introduced by [**CascudoCXY18**]. As mentioned before, we seek injective mappings from small fields to large fields, to enable techniques similar to those developed in [**BootleCG20**] and [**BootleCL22**] and construct an IOP protocol. More specifically, we use an efficient embedding that allows us to encode a elements in $\mathbb{F}_p$ into one element in $\mathbb{F}_{p^b}$, where $b = O(a)$. Then we construct IOPs with tensor queries for R1CS over $\mathbb{F}_p$, which make black-box use of tensor IOPs for the encoded R1CS over $\mathbb{F}_{p^b}$. One thing to be careful about is that operations in the large field are more expensive than those in the original small field. Thus, in order to keep our IOP efficient, we need to precompute the expensive operations, as we will describe in more detail in Section 2.5.

Now we discuss what characteristics we want our RMFEs to possess. First, they should be efficiently encodable to ensure the efficiency of the IOP where they will be used. Moreover, we want them to be systematic. In particular, given a basis of the large field $\mathbb{F}_{p^b}$, we think of the element as a vector of length b, where each entry in the vector is the coefficient of the i-th basis vector. We say that an RMFE is systematic if the coefficient vector of the encoded element is the original element padded with zeros. This property ensures that the tensor IOP has a linear-time prover. Lastly, we want the embeddings to preserve the component-wise additions and products of the original vectors in the small field $\mathbb{F}_p^a$ so that relations such as $\langle x, y \rangle$ are preserved. Keeping these properties in mind, we now present the formal definition of $(a,b)_p$-RMFE given in [**CascudoCXY18**] and also our definition of systematic $(a,b)_p$-RMFE.

Definition 2.1 ([**CascudoCXY18**]). *Let p be a prime power and $\mathbb{F}_p$ a field with p elements. Let $a, b \geq 1$ be integers. A pair (ϕ, ψ) is called a $(a,b)_p$-**reverse-multiplication-friendly embedding** (RMFE) if $\phi\colon \mathbb{F}_p^a \to \mathbb{F}_{p^b}$ and $\psi\colon \mathbb{F}_{p^b} \to \mathbb{F}_p^a$ are both $\mathbb{F}_p$-linear maps such that for all $\vec{x}, \vec{y} \in \mathbb{F}_p^a$,*

$$\vec{x} \circ \vec{y} = \psi(\phi(\vec{x}) \cdot \phi(\vec{y})) \ .$$

Note that this definition ensures that RMFEs preserve component-wise additions and products, as described in the previous paragraph. Therefore, in our later discussion, we on efficient encoding and the systematic property.

Definition 2.2. *An $(a, b)_p$-RMFE (ϕ, ψ) is* **systematic** *if there exists a basis $\{e_j\}_{j\in[b]}$ of the $\mathbb{F}_p$-linear space $\mathbb{F}_{p^b}$ such that for every element $\vec{x} = (x_1, \dots, x_a) \in \mathbb{F}_p^a$ it holds that $\phi(\vec{x}) = (x_1, \dots, x_a, 0, \dots, 0)$, where the i-th entry of $\phi(\vec{x})$ is the coefficient with respect to the i-th basis vector e_i.*

Note that this definition implies that systematic RMFE encodes a vector $\vec{x}$ over $\mathbb{F}_p$ into an element in $\mathbb{F}_{p^b}$ with $\|\vec{x}\|_0$ non-zero coefficients in basis E. Therefore, the cost of writing down the encoded vector is linear in terms of the input vector, which we can afford.

Warmup: systematic RMFE based on polynomial interpolation. We outline a simple RMFE construction based on polynomial interpolation from [**CascudoCXY18**], and explain how to make it systematic.

Fix k distinct elements $(\alpha_1, \dots, \alpha_k)$ in $\mathbb{F}_p$ and $\zeta \in \mathbb{F}_{p^{2k-1}}$ such that $\mathbb{F}_{p^{2k-1}} = \mathbb{F}_p(\zeta)$. The encoding map ϕ_{int} works a follows: on input $(x_1, \dots, x_a) \in \mathbb{F}_p^a$, compute the polynomial f over $\mathbb{F}_p[X]_{<k}$ such that $f(\alpha_i) = x_i$ for all $1 \le i \le k$, and output $f(\zeta)$. The decoding map ψ_{int} takes an element β in $\mathbb{F}_{p^{2k-1}}$ and determines the unique polynomial f in $\mathbb{F}_p[X]_{<2k-1}$ such that $\beta = f(\zeta)$ and outputs $(f(\alpha_1), \dots, f(\alpha_k))$. Intuitively, this forms an RMFE because the image of ϕ_{int} is the set of elements $\beta = f(\zeta) \in \mathbb{F}_{p^{2k-1}}$ where $\deg f \le a - 1$, which implies that the product of $f(\zeta), g(\zeta) \in \text{Im}\phi_{\text{int}}$ is represented as $(f \cdot g)(\zeta)$ since $\deg f \cdot g < 2k - 1$. Therefore, applying ψ_{int} will evaluate $f \cdot g$ at α_i as desired. Both maps are efficiently computable because polynomial interpolation is efficient.

Is it systematic? This RMFE is, in general, not systematic. For example, consider the canonical basis $\{1, \zeta, \dots, \zeta^{k-1}, \dots, \zeta^{(2k-1)-1}\}$ for $\mathbb{F}_{p^{2k-1}}$. Assume for the sake of contradiction that this basis satisfies the systematic property. Then following Definition 2.2, it must be that $\phi'(\alpha) = (\alpha_1, \dots, \alpha_a, 0, \dots, 0) = \sum_{i=1}^{a} \alpha_i \zeta^{i-1}$ for $\alpha = (\alpha_1, \dots, \alpha_a)$. However, according to the construction, we know that $\phi(\alpha) = \sum_{i=1}^{k} \alpha_i L_i(\zeta)$ where $L_i(\zeta) = \Pi_{\substack{1\le j\le k \\ j\ne i}} \frac{\zeta-\alpha_j}{\alpha_i-\alpha_j}$ is the i-th Lagrange polynomial. One can find some $\alpha \in \mathbb{F}_p^a$ such that $\phi'(\alpha) \ne \phi(\alpha)$, which shows that this basis does not give us the systematic property.

Making it systematic. Suppose that we change the canonical basis such that the first k elements are replaced by the Lagrange basis. Then each element in the image of the encoding can be represented as a linear combination of the first k elements in the basis, and the coefficient vector in the linear combination is exactly the input vector. With this change, the RMFE based on polynomial interpolation achieves the systematic property.

Why the above RMFE does not suffice. The RMFE based on polynomial interpolation requires the base field $\mathbb{F}_p$ to have size at least k (which can be improved to size at least $k - 1$ as in Section 4.1) as there have to be enough distinct elements for the interpolation. However, in this paper we will need to use embeddings for super-constant k and constant-sized base fields $\mathbb{F}_p$. Therefore we cannot use this RMFE based on polynomial interpolation.

A systematic RMFE based on AG codes. In this paper we use an RMFE based on algebraic geometry (AG) codes from [**CascudoCXY18**], which we show can be made systematic. Informally, instead of using elements of a finite field, the RMFE based on AG codes uses rational places in algebraic function fields. The embedding outputs the evaluation of the polynomial uniquely determined by the input at a place with high degree, which is similar to the case in the previous construction where it takes the value of the Lagrange-interpolated polynomial at α. In the more detailed explanation of this construction in Definition 4.6, we use tools from algebraic geometry, such as divisors, Riemann–Roch spaces, Ihara's constant, etc. to justify the properties we need. We postpone the full construction and detailed proof to Section 4.

2.4 A tensor IOP for R1CS over small fields

We use RMFEs to construct a tensor-query IOP for R1CS over any field. Below we summarize the result and construction, leaving technical details for Section 6.

Definition 2.3 (informal)**.** *A* $(\mathbb{F}, k, t)$**-tensor IOP** *is an IOP in which the prover sends* $\Pi_i \in \mathbb{F}^{\ell_i \cdot k^t}$ *for some positive integer* ℓ_i *and the verifier queries* $\langle v_0 \otimes v_1 \otimes \cdots \otimes v_t, \Pi_i \rangle$ *for* $v_0 \in \mathbb{F}^{\ell_i}$ *and* $v_1, \dots, v_t \in \mathbb{F}^k$.

Theorem 2.4 (informal)**.** *Let* $\phi \colon \mathbb{F}_p^a \to \mathbb{F}_{p^b}$ *be an RMFE, where* $b = O(a)$ *and the finite field* $\mathbb{F}_{p^b}$ *has size* $p^b \in \Omega(\log N)$. *There exists a* $(\mathbb{F}_{p^b}, k, t)$*-tensor IOP for* R1CS *instances of size* N *with* M *non-zero entries over* $\mathbb{F}_p$ *with the parameters below:*
- *soundness error is* $O(1)$*;*
- *round complexity is* $O(\log(N/a))$*;*
- *proof length is* $O(N/a)$ *elements in* $\mathbb{F}_{p^b}$*;*
- *query complexity is* $O(1)$*;*
- *the prover sends* $O(\log(N/a))$ *elements in* $\mathbb{F}_{p^b}$ *as non-oracle messages;*
- *the prover uses* $O(M + N)$ $\mathbb{F}_p$*-operations and* $O(tN/a)$ $\mathbb{F}_{p^b}$*-operations;*
- *the verifier uses* $O(M + N)$ $\mathbb{F}_p$*-operations and* $O(tk)$ $\mathbb{F}_{p^b}$*-operations;*
- *the verifier has randomness complexity* $O(\log(N/a))$ *over* $\mathbb{F}_{p^b}$ *and* $O(\log(N/a))$ *over* $\mathbb{F}_p$.

Prior constructions of tensor-query IOPs work only for large enough fields [**BootleCGGHJ17**; **BootleCG20**; **BootleCL22**]. Briefly, the use of the sumcheck protocol introduces a soundness error of $\Omega(\frac{\log N}{|\mathbb{F}|})$; moreover, holographic techniques to achieve sublinear verification introduce an even larger soundness error of $\Omega(\frac{N}{|\mathbb{F}|})$.

Straightforwardly expressing an R1CS problem over $\mathbb{F}_p$ as an R1CS problem over a large enough extension field leads to instances of more than linear size, and therefore does not suffice for our goal of linear-time IOPs. We show that, nevertheless, a more refined *embedding approach* does suffice.

Starting point: embedding R1CS from [CascudoG21]. Cascudo and Giunta [**CascudoG21**] rely on RMFEs to construct a linear-size IOP based on the Reed–Solomon code for R_{R1CS} over $\mathbb{F}_p$ for $p = 2$. Their IOP is not linear-time due to the use of the Reed–Solomon code. Yet their approach directly inspires our linear-time tensor IOP construction, and so we summarize their approach here as a starting point, generalised from $p = 2$ to any p.

The idea in [**CascudoG21**] is to use an RMFE embedding ϕ to embed the coefficient matrices A, B, C and the witness vector z into a field $\mathbb{F}_{p^b}$ where $p^b = \Omega(\log(N))$, and then show that $\phi(A)\phi(z) \circ \phi(B)\phi(z) = \phi(C)\phi(z)$, where ϕ is applied to matrices row-wise (so that the dimensions of the matrix multiplications match). We call this new problem over the large field *modular* R1CS. Then one can construct an IOP for modular R_{R1CS} over the large field $\mathbb{F}_{p^b}$. To make this reduction work, the prover additionally needs to prove relations of the form $\widetilde{x} \in \operatorname{Im}(\phi)$ for $\widetilde{x}$ that is claimed to be an embedded vector. Since ϕ is a linear function, [**CascudoG21**] introduces a subprotocol called modular lincheck to prove statements of the form $\widetilde{A}\widetilde{x} \in V^{\lambda}$ where V is a $\mathbb{F}_p$-linear subspace of $\mathbb{F}_{p^b}$ and λ is the dimension of the resulting vector being checked. We will now explain this subprotocol in more detail.

The modular lincheck in [CascudoG21]. Consider a matrix $A \in \mathbb{F}_p^{N \times N}$ with M non-zero entries, a vector $x \in \mathbb{F}_p^N$, a $(a, b)_p$-RMFE (ϕ, ψ), and a $\mathbb{F}_p$-linear subspace $V \subseteq \mathbb{F}_{p^b}$. Let $\widetilde{A} \in \mathbb{F}_{p^b}^{N \times N/a}$ and $\widetilde{x} \in \mathbb{F}_{p^b}^{N/a}$ be the alleged embeddings of A and x. The modular lincheck protocol checks whether $\widetilde{A}\widetilde{x} \in V^N$.

The usual approach to testing a linear relation of the form $Ax = b$ is *linear hashing*: the IOP verifier samples a random matrix R and then asks the prover to show that $R(Ax - b) = 0$. In the context of a modular lincheck, the IOP verifier samples a random matrix $R \in \mathbb{F}_{p^b}^{\lambda \times N}$. It first checks that $R\widetilde{A}\widetilde{x} = y$ for some $y \in \mathbb{F}_{p^b}^{\lambda}$ using a lincheck protocol adapted from the one in [**BenSassonCRSVW19**] with input matrix $R\widetilde{A}$ and vector $\widetilde{x}$ and designated matrix-vector product y. We define ϵ_{LC} to be the soundness error of this check. Then the verifier checks that $y \in V^{\lambda}$. Since V is a $\mathbb{F}_p$-linear space, R needs to be sampled from $\mathbb{F}_p^{\lambda \times N}$ to ensure that $R\widetilde{A}\widetilde{x} \in V^{\lambda}$ whenever $\widetilde{A}\widetilde{x} \in V^N$.

The linear hashing matrix $R \in \mathbb{F}_p^{\lambda \times N}$ in [**CascudoG21**] is sampled as follows. Sample an element $\alpha \in \mathbb{F}_{p^\lambda}$ uniformly at random. The field $\mathbb{F}_{p^\lambda} \cong \mathbb{F}_p[X]/(P)$ for some irreducible polynomial P over $\mathbb{F}_p$, and every element $f \in \mathbb{F}_p[X]/(P)$ has a matrix representation $M_f \in \mathbb{F}_p^{\lambda \times \lambda}$ such that for any $g = \sum_{i=0}^{\lambda-1} a_i X^i \in \mathbb{F}_p[X]/(P)$ with associated coefficient vector $\vec{g}$, $M_f \cdot \vec{g}$ is the coefficient vector for $f \cdot g \in \mathbb{F}_p[X]/(P)$. The matrix R is then defined by horizontally concatenating the matrices $M_\alpha, M_{\alpha^2}, \ldots, M_{\alpha^N}$.

This choice of R guarantees that this check has soundness error bounded by $\frac{N/\lambda}{p^\lambda}$. Here N/λ is the highest degree of α that is used to generate R and p^λ is the order of the field that α is sampled from. Thus, the overall modular lincheck protocol has soundness error $\frac{N/\lambda}{p^\lambda} + \epsilon_{\mathrm{LC}}$.

Finally we note that the modular lincheck protocol in [**CascudoG21**] is a Reed–Solomon encoded IOPP. Namely the verifier's oracle is encoded by a Reed–Solomon code, and the oracle is accessed via point queries. This protocol can be converted to a tensor IOP by observing that a point query to a Reed–Solomon encoding of Π can be simulated by a tensor query to Π. Thus we can extract a tensor IOP to test the same relation.

Our construction. It is tempting to directly combine the modular lincheck protocol from [**CascudoG21**] and the linear-time tensor IOP from [**BootleCG20**] to get a linear-time tensor IOP for R_{R1CS} over $\mathbb{F}_p$. This, however, does not work. We describe the main problems that arise and how we solve them.

Problem 1: computing $\phi(A)$ and $R\phi(A)$ takes superlinear time. For certain instance matrices A, a naive $(a, b)_p$-RMFE ϕ would make the prover's time superlinear. Consider a matrix $A \in \mathbb{F}_p^{N \times N}$ with M non-zero entries, and recall that ϕ is applied to A row-wise. If any two non-zero elements in the same row of A are a-entries apart from each other, then $\phi(A) \in \mathbb{F}_{p^b}^{N \times N/a}$ also has M non-zero elements in $\mathbb{F}_{p^b}$. The embedding takes a matrix A with M elements in $\mathbb{F}_p$ to a matrix $\phi(A)$ with M elements in $\mathbb{F}_{p^b}$. If we naively write down $\phi(A)$ as elements in $\mathbb{F}_{p^b}$, this step alone would take $O(M)$ $\mathbb{F}_{p^b}$-operations which could cost at least bM $\mathbb{F}_p$-operations. Since $\left|\mathbb{F}_{p^b}\right| = \Omega(\log N)$, this step incurs superlinear cost in M.

Furthermore, passing $\phi(A)$ to the tensor IOP for the modular R_{R1CS} over $\mathbb{F}_{p^b}$ would result in the prover having arithmetic complexity $\Theta(M)$ over $\mathbb{F}_{p^b}$: in the modular lincheck protocol the prover computes the product $R\phi(A)$ (for $R \in \mathbb{F}_p^{\lambda \times N}$ and $\phi(A) \in \mathbb{F}_{p^b}^{N \times N/a}$) which takes λM operations over $\mathbb{F}_{p^b}$. This step is also superlinear in M over $\mathbb{F}_p$ when $\left|\mathbb{F}_{p^b}\right| = \Omega(\log N)$.

Solution: systematic RMFEs. We rely on a systematic RMFE with basis E (discussed in Section 2.3) in order to write down $\phi(A)$ and compute $R\phi(A)$ in linear time. First we represent every element in $\mathbb{F}_{p^b}$ in the basis E. Recall that E is a basis for the $\mathbb{F}_p$-linear space $\mathbb{F}_{p^b}$. For any field element $f \in \mathbb{F}_{p^b}$ we use $\mathsf{E}(f) \in \mathbb{F}_p^b$ to denote f's coefficient vector under E, and for any matrix H in $\mathbb{F}_{p^b}^{x \times y}$ we use $\mathsf{E}(H) \in \mathbb{F}_p^{x \times by}$ to denote the representation of H such that each entry is encoded by the corresponding coefficient vector under E.

Since the RMFE ϕ is systematic with respect to E, for any matrix A with M non-zero entries, $\mathsf{E}(\phi(A))$ also has M non-zero entries. Thus writing down $\mathsf{E}(\phi(A))$ takes $O(M)$ $\mathbb{F}_p$-operations.

Additionally, since $\mathbb{F}_{p^b}$ is $\mathbb{F}_p$-linear, for every $f \in \mathbb{F}_{p^b}$ and $e \in \mathbb{F}_p$, $\mathsf{E}(ef) = e\mathsf{E}(f)$ where the multiplication of e by $\mathsf{E}(f)$ is performed over $\mathbb{F}_p$. Then computing $\mathsf{E}(R\phi(A)) = R\mathsf{E}(\phi(A))$ requires only λM operations over $\mathbb{F}_p$. For addition of two elements over the large field, as discussed in Section 2.3, given a vector $v \in \mathbb{F}_p^a$, the given systematic RMFE would encode it into an element in $\mathbb{F}_{p^b}$ with $\|v\|_0$ non-zero coefficients in basis E. Therefore, we only need to do a linear number of additions of the coefficients to add two elements over the large field. Each addition requires a linear number of operations, which is not problematic.

Problem 2: super constant $\boldsymbol{\lambda}$. Now that computing $R\phi(A)$ takes $O(\lambda M)$ $\mathbb{F}_p$-operations, λ must be a constant for the prover and the verifier to run in time linear in M. However, in the Boolean case, as analysed in the modular lincheck protocol above, λ needs to be at least $\log N$ for the soundness error $\frac{N}{\lambda p^\lambda}$ to be non-trivial.

Solution: subspace biased generators. We use an alternative random hashing matrix. We conceptualize the linear hashing step, and come up with a useful definition of a (p, λ)-subspace ϵ_{sub}-biased generator G_{sub}, which generates random hashing matrices R such that for any $\mathbb{F}_p$-linear subspace $V \subseteq \mathbb{F}_{p^b}$ and $z \notin V^N$, we have $\Pr_{R \leftarrow G_{\mathsf{sub}}(\mathbb{F}^s_{p^\lambda})}[Rz \in V^\lambda] \leq \epsilon_{\mathsf{sub}}$. This definition fully captures the required property of the linear hashing matrix. We also provide a generic construction of a (p, λ)-subspace ϵ_{sub}-biased generator for any constant integers p and λ, and any constant soundness error $\epsilon_{\mathsf{sub}} > \frac{1}{p^\lambda}$ that additionally has arithmetic complexity $O_{\lambda,p,\epsilon_{\mathsf{sub}}}(N)$. So replacing the random R in [**CascudoG21**] with the output of such a generator G_{sub} ensures that λ is a constant and that the soundness error is non-trivial.

2.5 Eliminating overhead through precomputation

In fact, the techniques described above generalise from binary fields to any finite field $\mathbb{F}_p$. Given a systematic $(a, b)_p$-RMFE (ϕ, ψ) with basis E, where $b = \eta \cdot a$ for some constant η, we can construct an IOP with prover arithmetic complexity $O(M)$ operations over $\mathbb{F}_p$ and $O(N/b)$ operations over $\mathbb{F}_{p^b}$. To offset the soundness error mentioned in Section 2.1, we have $b \in \Omega(\log \log N)$.

Our main theorem requires prover arithmetic complexity $O(M)$ operations over $\mathbb{F}_p$. Consider the cost of $O(N/b)$ operations over $\mathbb{F}_{p^b}$ measured over $\mathbb{F}_p$. Clearly, *addition* operations over $\mathbb{F}_{p^b}$ costs $O(b)$ operations over $\mathbb{F}_p$, regardless of the $\mathbb{F}_p$-basis used to represent elements of $\mathbb{F}_{p^b}$. Therefore, $O(N/b)$ operations over $\mathbb{F}_{p^b}$ translates to $O(M)$ operations over $\mathbb{F}_p$.

However, since b is super-constant, each *multiplication* operation over $\mathbb{F}_{p^b}$ costs a superlinear (in b) number of basic operations over $\mathbb{F}_p$. For example, in the standard basis, it costs $O(b^2)$ operations in $\mathbb{F}_p$ to multiply x_1 and $x_2 \in \mathbb{F}_{p^b}$ using a schoolbook multiplication algorithm. Since the IOP described above uses $O(N/b)$ operations over $\mathbb{F}_{p^b}$, this translates to $O(bN)$ prover operations over $\mathbb{F}_p$, which could be as much as $N \log \log N$! Even the best known multiplication algorithm by [**HarveyH21**][1] requires $O(b \log b)$ operations, which would lead to $N \log^3 N$ prover operations. In fact, [**AfshaniFKL19**] gives evidence that multiplication in $O(b \log b)$ operations is optimal under a central conjecture in network coding theory; any improvement over this bound would be a breakthrough result. Therefore, we cannot hope for our tensor IOP to achieve linear prover complexity simply by using better algorithms over $\mathbb{F}_{p^b}$.

Precomputing multiplication tables. We address this problem by observing that we can afford to precompute *every possible operation* over $\mathbb{F}_{p^b}$ in $O(N)$ operations over $\mathbb{F}_p$, if b is *sufficiently small*. Then, instead of performing each of the $O(N/b)$ operations over $\mathbb{F}_{p^b}$ from scratch, one can simply retrieve the result of each operation over $\mathbb{F}_{p^b}$ from a precomputed look-up table. We formalize this approach by modeling algorithms as random-access machines, and providing straightforward statements about precomputation.

The multiplication table for $\mathbb{F}_{p^b}$ is a matrix of size $p^b \times p^b$. Computing each entry of the matrix costs $O(b^2)$ operations over $\mathbb{F}_p$ via the naive multiplication algorithm, leading to a total precomputation cost of $O(b^2 p^{2b})$ operations over $\mathbb{F}_p$. Afterwards, the result of any multiplication over $\mathbb{F}_{p^b}$ can be loaded from the lookup table stored in memory in $O(1)$ operations.

[1]This is an algorithm for integer multiplication

For $b = \frac{1}{2}\log N - o(\log N)$, we have $O(b^2 p^{2b}) = O(N)$, giving a linear time complexity. Note that this places a strict upper limit of $\mathbb{F}_{p^b} < \sqrt{N}$ on the size of the extension fields used in our tensor IOP.

The technical details for precomputation for RAM are in Section 9.

2.6 Algebraic automata

The previous sections strive to optimize the running time of the prover, but verification time is also an important efficiency parameter that we want to improve. Although we do not know how to improve verification time for general R1CS instances, we explain how to achieve sublinear verification for algebraic automata.

From algebraic automata to R1CS. Since an algebraic automata instance is defined in a similar way to an R1CS instance, we are able to reduce an algebraic automata instance to a special R1CS instance. Let M be a matrix in $\{A, B, C\} \subset \mathbb{F}^{w \times 2w}$ and $z : [T+1] \to \mathbb{F}^w$. Definition 1.4 computes $M(z(t), z(t+1))$ for every computation time step t. If we view z as a vector in $\mathbb{F}^{w(T+1)}$, right-multiplying z by a large matrix S_M in which the diagonal entries are T repeated blocks of M gives a tall vector with all desired $M(z(t), z(t+1))$. More formally, $S_M = [M, \ldots, M]$ is a diagonal matrix of dimension $wT \times w(T+1)$. Therefore, we could rewrite the algebraic automata constraints as $S_A z \circ S_B z = S_C z$, which is an R1CS instance where the matrices have special structure.

Holographic tensor IOP for algebraic automata. We previously described a tensor IOP for general R1CS instances. The tensor IOP has a linear-time verifier due to the subprotocol checking $\phi(A)\vec{x} = \vec{x}_A$. The general approach to checking equations of this form is to check that for some random vector r, $r^\intercal \phi(A)\vec{x} = r^\intercal \vec{x}_A$. While the right hand side can be verified efficiently by using relatively standard techniques, computing the left hand side naively could take time linear in the number of non-zero entries in $\phi(A)$. One general strategy taken by [**BootleCG20**] is to rewrite the quadratic form as

$$r^\intercal \phi(A)\vec{x} = \sum_i r(\mathrm{row}_A(i))\mathrm{val}_A(i)\vec{x}(\mathrm{col}_A(i)),$$

where val_A is a vector of all the nonzero entries in $\phi(A)$, row_A is the vector of their row indices, and col is the vector of their column indices. Given the vectors $r(\mathrm{row}_A(*))$, val_A, and $\vec{x}(\mathrm{col}_A(*))$ the verifier can efficiently check the value of the right hand side via standard procedures. The hard part is to check that $r(\mathrm{row}_A(*))$ and $\vec{x}(\mathrm{col}_A(*))$ are correctly generated given r and $\vec{x}$. Previous works [**BootleCG20**; **Bootle2022**] have used look-up protocols to check this relation. However those protocols crucially rely on $\Omega(n)$ size fields. We cannot afford to embed into such large fields due to the overhead in the precomputation phase. So instead, we look at the special case of staircase matrices $\phi(A)$ for which $r(\mathrm{row}_A(*))$ and $\vec{x}(\mathrm{col}_A(*))$ are well-structured. Essentially, when $\phi(A)$ is a staircase matrix, it can be decomposed into two almost-diagonal matrices $\phi(A)_0 + \phi(A)_1$. Their corresponding row index vectors satisfy $r(\mathrm{row}_0(*)) = r(\mathrm{row}_1(*)) = r([1, \ldots, n]) = r$, and their column index vectors satisfy $\vec{x}(\mathrm{col}_0(*)) = \vec{x} = \mathrm{shift}(\vec{x}(\mathrm{col}_1(*)))$. Specifically, $\mathrm{col}_0(*) = [1, 2 \ldots, n-1]$ and $\mathrm{col}_0(*) = [2, 3 \ldots, n]$. So we design a cyclic shift protocol to check this shift relation for instances over small fields.

Shift protocol over small fields. We require a tensor IOP protocol for the following relation.

Definition 2.5. *The* **shift** *relation $R_{\circlearrowright}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, s, N), (a, b))$$

where $N = k^t$, $b \in \mathbb{F}^N$ is the cyclic shift *of $a \in \mathbb{F}^N$, which we denote $b = \mathrm{shift}(a)$, if for all $i \in [N - s]$, it holds that $a_i = b_{i+s}$, and for $i \geq N - s$, $a_i = 0$.*

We explain how the new protocol works, using the shift protocol in [**BootleCG20**] as a starting point. The shift protocol in [**BootleCG20**] (for $s = 1$) considers the polynomials $A(X) := \sum_{i=1}^{N} a_i X^{i-1}$ and $B(X) := \sum_{i=1}^{N} b_i X^{i-1}$, and uses the fact that $X \cdot A(X)$ is approximately equal to $B(X)$, up to some small corrections involving a constant number of coefficients of $A(X)$ and $B(X)$. This follows from the fact that if $a = \text{shift}(b)$, then $a_i = b_{i+1}$ for $i \leq N-1$, and so the i-th coefficient of $XA(X) - B(X)$ is equal to zero.

This leads to a test for $R_{\circlearrowright}$, checking that a polynomial identity of degree N is satisfied at a random point $\gamma \in \mathbb{F}$. By the Schwartz–Zippel lemma, if $b \neq \text{shift}(a)$, then the probability that $A(\gamma) - \gamma \cdot B(\gamma) = (1 - \gamma^N) \cdot b_N$ is at most $N/|\mathbb{F}|$.

This polynomial identity test relies on the structure of monomials X^i and the ability to scale two related polynomials so that their coefficients match. However, we cannot use it in our setting since the polynomial identity used has degree N, which may be larger than the number of elements in the finite field $\mathbb{F}$, leading to a trivial bound on the soundness error of the protocol. Therefore, we design a more complex solution.

Suppose that $N = k^t$. In the tensor IOP setting, we can evaluate the polynomials $A(X)$ and $B(X)$ at any point $\gamma \in \mathbb{F}$ using a tensor query:

$$(1, \gamma, \gamma^2, \ldots, \gamma^{k-1}) \otimes (1, \gamma^k, \gamma^{2k, \ldots, \gamma^{k(k-1)}}) \otimes \cdots \otimes (1, \gamma^{k^{t-1}}, \gamma^{2k^{t-1}}, \ldots, \gamma^{k^{t-1}(k-1)}) \ .$$

However, as discussed, this leads to a polynomial identity of degree $N = k^t$. A natural way to reduce the polynomial dimension of this tensor query is to consider the following alternative query for random points $\gamma_1, \ldots, \gamma_t \in \mathbb{F}$:

$$(1, \gamma_1, \gamma_1^2, \ldots, \gamma_1^{k-1}) \otimes (1, \gamma_2, \gamma_2^2, \ldots, \gamma_2^{k-1}) \otimes \cdots \otimes (1, \gamma_t, \gamma_t^2, \ldots, \gamma_t^{k-1}) \ . \tag{1}$$

This query provides evaluations of the polynomial $A(X_1, \ldots, X_t) := \sum_{i_1, \ldots, i_t \in \{0, \ldots, k-1\}} a_{i_t, \ldots, i_1} X_1^{i_1} \ldots X_t^{i_t}$, and similarly for $B(X)$, where we have reindexed the $N = k^t$ entries of a using a k-ary representation. However, it is no longer possible to scale $A(X_1, \ldots, X_t)$ by a single monomial as in the univariate case in order to get a polynomial approximately equal to $B(X_1, \ldots, X_t)$.

In fact, the appropriate scaling factor now differs according to the k-ary representation of i. Let $s < k$, and consider the number of carries when adding s to $(i_t, \ldots, i_1)$.

- If $0 \leq i_1 \leq k-1-s$, then there are no carries when adding s to the k-ary representation $(i_t, \ldots, i_1)$. This means that $a_{i_t, \ldots, i_1}$ is equal to $b_{i_t, \ldots, i_1+s}$. Since $a_{i_t, \ldots, i_1}$ and $b_{i_t, \ldots, i_1+s}$ appear as coefficients of monomial $X_1^{i_1} \ldots X_t^{i_t}$ when a and b are queried using the tensor query from Equation (1), these coefficients can be related using the scaling factor X_1^s.

- If $k-s \leq i_1 \leq k-1$, and $0 \leq i_2 \leq k-2$, then there is one carry when adding s to the k-ary representation $(i_t, \ldots, i_1)$. This means that $a_{i_t, \ldots, i_1}$ is equal to $b_{i_t, \ldots, i_2+1, i_1+s-k}$. Since $a_{i_t, \ldots, i_1}$ and $b_{i_t, \ldots, i_2+1, i_1+s-k}$ appear as coefficients of monomial $X_1^{i_1} \ldots X_t^{i_t}$ when a and b are queried using the tensor query from Equation (1), these coefficients can be related using the scaling factor $X_1^{s-k} X_2$.

Using similar reasoning to the above, it is possible to determine a suitable scaling factor for each possible number of carries from 1 up to t. By setting certain coefficients of the tensor query in Equation (1) to zero, it is possible to make queries which only involve the entries of a and b involved in the r-carry case. The verifier must make a tensor query to a and b in each case, and multiply by scaling factors. This leads to a total of $O(t)$ queries. Furthermore, the maximum degree of polynomials used in the protocol is now $O(tk)$, meaning that by setting t large enough, we can use the shift protocol over small fields.

Checking the output of the biased generator. To achieve sublinear verification, we also need to find a way to check the output of the biased generator efficiently. More specifically, in the protocol for general

R1CS described in Section 2.4, we described the use of linear-time-computable subspace biased generators. However, to achieve sublinear verification, the verifier cannot afford to naively check whether the output of the subspace biased generator is consistent with the randomness chosen by the verifier. On the other hand, it is possible to efficiently check that a vector has a tensor structure using protocols similar to the tensor-consistency test in [**BootleCG20**], which has sublinear verification. Therefore, we will design our subspace biased generator with tensor structure in mind.

Our subspace biased generator is constructed from ϵ-biased generators. Given an ϵ-biased generator $G\colon \mathbb{F}_{p^\lambda}^s \to \mathbb{F}_{p^\lambda}^{b'}$, let $G'(r_1,\ldots,r_t) \coloneqq G(r_1)\otimes\cdots\otimes G(r_t)$ for $r_1,\ldots,r_t \in \mathbb{F}_{p^\lambda}^s$. Note that G' is a $t\epsilon$-biased generator. We then define the corresponding subspace biased generator to be the matrix representation of G'. Let $r = (r_1,\ldots,r_t) \in \left(\mathbb{F}_{p^\lambda}^s\right)^t$. Note that $G'(r)$ can be viewed as a matrix in $\mathbb{F}_p^{\lambda\times\lambda}$ for all i, and we define $G_{\mathsf{sub}}(r)_i$ to be this matrix. Viewed as a vector of field-extension elements, the biased generator output has a tensor structure, which would be easy to check using the tensor-consistency test in [**BootleCG20**]. However, we require the matrix representation of G' instead as parts of our protocol for algebraic automata use calculations involving individual elements of the base field. The proof that this construction gives a subspace biased generator with the desired property is explained more carefully in Section 5.1. Further, in our real protocol, we use a slightly more complicated biased-generator construction, for reasons explained in Section 7.

Now we can think of the output of our subspace biased generator $G_{\mathsf{sub}}\colon \left(\mathbb{F}_{p^\lambda}^s\right)^t \to \mathbb{F}_p^{\lambda\times\lambda b'^t}$ as b'^t matrices in $\mathbb{F}_p^{\lambda\times\lambda}$. Define $y^{(m_0,m_t)} \in \mathbb{F}_p^{b'^t}$ to be such that its k-th entry is the (m_0,m_t)-th entry of the k-th matrix of $G_{\mathsf{sub}}(r)$. We can write $y^{(m_0,m_t)}$ for all $m_0,m_t \in [\lambda]$ as a sum of tensor products because of the tensor structure of G' and the properties of matrix multiplication. A detailed calculation is presented in Section 7.4. Then, we successfully reduce the task of checking that the output of a subspace biased generator was computed correctly, to checking that the output is a sum of tensor products, which can be done by modifying an established protocol, the tensor-consistency test, in [**BootleCG20**] and meet our constraints on verifier efficiency.

Bottleneck for improving on sublinear verification. With this approach, the bottleneck for verifier efficiency is checking that the output of the biased generator was computed correctly. As explained above, in this subprotocol the verifier uses $O(t\cdot \mathsf{o}_G)$ operations where o_G is the time complexity of computing the ϵ-biased generator $G\colon \mathbb{F}_{p^\lambda}^s \to \mathbb{F}_{p^\lambda}^{b'}$. By Corollary 5.10, we can construct such a biased generator for any $\epsilon > \frac{1}{p^\lambda}$ and time complexity $\mathsf{o}_G = \Theta(b')$ $\mathbb{F}_{p^\lambda}$-operations. Since we need to pick $\lambda \in O(1)$ in order to achieve linear prover time for the IOP protocol, $\epsilon \in \Omega(1)$. Therefore in order to obtain a nontrivial tensor-structured $t\epsilon$-biased generator G', we need to pick the tensor rank $t < \frac{1}{\epsilon} < p^\lambda$. For such choices of t, the verifier time for checking the output of G' becomes $\Theta(t\cdot\mathsf{o}_G) = \Theta(t\cdot(wT)^{1/t})$ $\mathbb{F}_{p^\lambda}$-operations which translates to $\Theta_{p,\lambda}((wT)^{1/t})$ operations over $\mathbb{F}_p$.

This approach cannot achieve polylogarithmic verification time because we do not know how to construct ϵ-biased generators over fields $\mathbb{F}$ with $\epsilon < \frac{1}{|\mathbb{F}|}$. This fact holds true even for the trivial biased generator that simply outputs the random seed. To summarize, there is a tradeoff between soundness and verification time but since we are working over the small field $\mathbb{F}_{p^\lambda}$, the best verification time that we can achieve is sublinear.

2.7 Completing the point-query IOP construction

We explain how to combine the ingredients described in previous sections to obtain the IOP in Theorem 1.2. The IOP described in Theorem 1.5 uses similar techniques.

From tensor queries to point queries. We transform the tensor-query IOP from Section 2.4 into a point-query IOP by following the strategy of [**BootleCG20**], which uses point queries to check the correctness of claimed answers to tensor queries. The IOPP in [**BootleCG20**] encodes each proof message Π in the tensor-query IOP using a suitable tensor code to obtain an encoded proof message $\hat{\Pi}$ for the point-query IOP; then the IOPP relies on a *proximity test* to ensure that each $\hat{\Pi}$ is close to a valid tensor encoding.

The proximity test in [**BootleCG20**] relies on the fact that taking random linear combinations of noisy codewords preserves their distance from the code up to some small error probability. Later, the proximity test of [**BootleCL22**] replaced purely random linear combinations with certain structured combinations that use a much smaller seed. Unfortunately, in either case, the error-probabilities associated with taking linear combinations contribute $O(N/|\mathbb{F}|)$ to the soundness error of the proximity test. This is unacceptable because as explained in Section 2.5, we can only apply precomputation techniques in fields of size $o(\sqrt{N})$.

To solve this problem, we generalize the [**BootleCG20**; **BootleCL22**] proximity tests by replacing the linear combinations with more general *proximity generators*, which are randomness generators whose outputs obey the same distance preservation property. We then instantiate the proximity generator with a different construction (given in Section 8.3) than those used in [**BootleCG20**; **BootleCL22**], which has better error probability.

However, when applied to the tensor IOP from Section 2.4, the new consistency test IOPP has query complexity $O(N^{1/t})$ which is insufficient for Theorem 1.2.

Interactive proof composition. We follow [**BootleCL22**], by using interactive proof composition to reduce the query complexity of the IOPP to $\mathsf{polylog}(N)$ over $\mathbb{F}_p$. This involves robustifying the IOPP via a linear code, and then applying the proof composition transformation in [**BenSassonCGRS17**] with the robustified consistency check as the outer IOP and the PCPP from [**Mie09**] as the inner IOP. This yields an IOPP with query complexity $\mathsf{polylog}(N)$.

When applied to the tensor IOP of Section 2.4, which is defined over $\mathbb{F}_{p^b}$, the resulting point-query IOP for R1CS over $\mathbb{F}_p$ is an IOP over $\mathbb{F}_{p^b}$, with query complexity $\mathsf{polylog}(N)$, and prover and verifier arithmetic complexity $O(M)$ operations in $\mathbb{F}_p$ and $O(N/a)$ multiplications in $\mathbb{F}_{p^b}$. This is still insufficient for Theorem 1.2 because the overhead of performing arithmetic operations over $\mathbb{F}_{p^b}$ leads to superlinear prover costs over $\mathbb{F}_p$.

Precomputing the multiplication table. We solve this problem using the precomputation techniques of Section 2.5, precomputing the multiplication table of $\mathbb{F}_{p^b}$. Each of the $O(N/a)$ multiplications over $\mathbb{F}_{p^b}$ can be replaced by looking up the product in the multiplication table. This means reading a single element of $\mathbb{F}_{p^b}$, which can be written as $O(a)$ elements of $\mathbb{F}_p$.

Thus, the $O(N/a)$ multiplications in $\mathbb{F}_{p^b}$ are replaced by $O(N)$ operations over $\mathbb{F}_p$, plus the $O(b^2p^{2b})$ operations required to precompute the multiplication table. Setting $b = \frac{1}{2}\log N - o(\log N)$ so that $O(b^2p^{2b}) = O(N)$, the $O(N)$ operations over $\mathbb{F}_p$ dominate, giving an IOP with both prover and verifier arithmetic complexity $O(M + N)$ over $\mathbb{F}_p$.

This gives a point-query IOP for R_{R1CS} over $\mathbb{F}_p$ with linear-time prover and verifier. The construction of the consistency test IOPP is given in Section 8. A formal explanation of precomputation techniques is given in Section 9, and Section 10 gives a rigorous analysis of the final IOP and its parameters.

3 Preliminaries

Definition 3.1. *An* **indexed relation** R *is a set of triples* $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ *where* $\mathbb{i}$ *is the index,* $\mathbb{x}$ *the instance, and* $\mathbb{w}$ *the witness. The corresponding* **indexed language** $L(R)$ *is the set of pairs* $(\mathbb{i}, \mathbb{x})$ *for which there exists a witness* $\mathbb{w}$ *such that* $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in R$.

Definition 3.2. *A* **IOP with query class** $\mathcal{Q}$ *(some set of functions) is a tuple* $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$*, where* $\mathbf{I}$ *is the* indexer*,* $\mathbf{P}$ *the* prover*, and* $\mathbf{V}$ *the* verifier*. The indexer is a deterministic polynomial-time algorithm, while the prover and verifier are probabilistic polynomial-time interactive algorithms.*

In an offline phase, the indexer $\mathbf{I}$ *is given an index* $\mathbb{i}$ *and outputs an encoding* Π_0 *of* $\mathbb{i}$.

In an online phase, the prover $\mathbf{P}$ *receives as input an index-instance-witness tuple* $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ *and the verifier* $\mathbf{V}$ *receives as input the instance* $\mathbb{x}$*; in addition, the verifier* $\mathbf{V}$ *has query access to* Π_0 *(in a precise sense specified below), which we denote as* $\mathbf{V}^{\Pi_0}(\mathbb{x})$*. The online phase consists of multiple rounds, and in each round the prover* $\mathbf{P}$ *sends a proof message* Π_i *and the verifier* $\mathbf{V}$ *replies with a challenge message* ρ_i.

The prover $\mathbf{P}$ *may compute its proof message* Π_i *based on its input* $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ *and all the verifier challenges received thus far (none if* $i = 1$ *or* $\rho_1, \ldots, \rho_{i-1}$ *if* $i > 1$*). In contrast, the verifier* $\mathbf{V}$ *may compute its challenge message* ρ_i *based on its input* $\mathbb{x}$ *and on answers obtained by querying* $(\Pi_0, \Pi_1, \ldots, \Pi_i)$ *via queries in* $\mathcal{Q}$*. In more detail, the answer of a query* $q \in \mathcal{Q}$ *to* $(\Pi_0, \Pi_1, \ldots, \Pi_i)$ *is* $v := q(\mathbb{x}, \Pi_0, \Pi_1, \ldots, \Pi_i)$ *(this answer could also be a special error value in case the proof messages are not according to an expected format).*

After the interaction and all queries are concluded, the verifier $\mathbf{V}$ *accepts or rejects.*

Remark 3.3 (non-oracle messages)**.** We allow the prover in an IOP to also send, at any point in the interaction, arbitrary messages that the verifier will simply read in full (without making any queries) as in a usual interactive proof. We refer to such messages as *non-oracle messages*, to differentiate them from the *oracle messages* to which the verifier has query access. These non-oracle messages can typically be viewed as degenerate cases of oracle messages, and we use them in protocol descriptions for convenience of exposition.

A holographic interactive oracle proof $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ for an indexed relation R has completeness 1 and soundness error ϵ if the following holds.

- **Completeness.** For every index-instance-witness tuple $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in R$, the probability that $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept is 1.
- **Soundness.** For every index-instance tuple $(\mathbb{i}, \mathbb{x}) \notin L(R)$ and malicious prover $\widetilde{\mathbf{P}}$, the probability that $\widetilde{\mathbf{P}}$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept is at most ϵ.

Public coins. A holographic IOP is *public-coin* if each verifier message to the prover is a random string. This means that the verifier's randomness is its challenge messages $\rho_1, \ldots, \rho_{\mathsf{rc}}$. All verifier queries can be postponed, without loss of generality, to a query phase that occurs after the interactive phase with the prover.

Non-adaptive queries. A holographic IOP is *non-adaptive* if all verifier queries depend solely on the input instance $\mathbb{x}$ and the verifier's randomness, as opposed to some queries depending on answers to prior queries. For non-adaptive IOPs, the verifier $\mathbf{V}$ can be written as a pair of algorithms $(\mathbf{V}_{\mathrm{q}}, \mathbf{V}_{\mathrm{d}})$ where: (a) $\mathbf{V}_{\mathrm{q}}$ is probabilistic, takes as input the instance $\mathbb{x}$, interacts with the prover $\mathbf{P}$, and outputs a decision state σ and a query set I (for the index oracle and proof oracles); and (b) $\mathbf{V}_{\mathrm{d}}$ is deterministic, takes as input the decision state σ and query answers $\mathbf{v} \in \Sigma^I$, and outputs a decision bit. For this case, we define the relation $R(\mathbf{V})$ to be all pairs $(\sigma, \mathbf{v})$ such that there exist $\mathbb{x}$ and I with (σ, I) in the support of $\mathbf{V}_{\mathrm{q}}(\mathbb{x})$ and $\mathbf{V}_{\mathrm{d}}(\sigma, \mathbf{v}) = 1$.

Complexity measures. We consider several complexity measures:

– *round complexity* rc is the number of back-and-forth message exchanges between the prover and verifier;
– *answer alphabet* Σ is the alphabet over which oracle messages are defined;
– *proof length* $\mathsf{l} = \mathsf{li} + \mathsf{lp} + \mathsf{lc}$ where $\mathsf{li} := |\Pi_0|$ is the number of alphabet symbols output by the indexer, $\mathsf{lp} := |\Pi_1| + \cdots + |\Pi_{\mathsf{rc}}|$ is the total number of alphabet symbols sent in oracle messages by the prover, and lc is the total number of alphabet symbols sent in non-oracle messages by the prover;
– *randomness* r is the number of random bits used by the verifier;
– *query complexity* q is the total number of queries made by the verifier (to any oracle);
– *running time* ti is the running time of $\mathbf{I}$, tp is the running time of $\mathbf{P}$, and tv is the running time of $\mathbf{V}$. In the non-adaptive case, tq and td are the running times of the query and decision components of $\mathbf{V}$ respectively.

We define the two query classes that we use in this paper, point queries and tensor queries.

Definition 3.4. *A* **holographic IOP with point queries** *is an IOP with the query class $\mathcal{Q}_{\text{point}}$ defined as follows: $\mathcal{Q}_{\text{point}}$ is all functions of the form $q(\mathbb{x}, \Pi_0, \Pi_1, \ldots, \Pi_i) = \Pi_j[k]$ for some $j \in \{0, 1, \ldots, i\}$ and location k. (If the location k does not exist, the answer is an error.) Namely, each query in the class $\mathcal{Q}_{\text{point}}$ returns the symbol at a location of the encoded index ($j = 0$) or of a specified prover message ($j > 0$).*

Definition 3.5. *Given a finite field $\mathbb{F}$ and positive integers k, t, a* **holographic IOP with $(\mathbb{F}, k, t)$-tensor queries** *is an IOP with the query class $\mathcal{Q}_{\text{tensor}}(\mathbb{F}, k, t)$ defined as follows: $\mathcal{Q}_{\text{tensor}}(\mathbb{F}, k, t)$ contains all functions of the form $q(\mathbb{x}, \Pi_0, \Pi_1, \ldots, \Pi_i) = \langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi_j \rangle$ for some $j \in \{0, 1, \ldots, i\}$ and vectors $q_0 \in \mathbb{F}^*$ and $q_1, \ldots, q_t \in \mathbb{F}^k$. (If the lengths of the linear combination $q_0 \otimes q_1 \otimes \cdots \otimes q_t$ and proof string Π_j do not match, the answer is an error.) Namely, each query in the class $\mathcal{Q}_{\text{tensor}}$ returns the scalar product of a certain tensor vector and the encoded index ($j = 0$) or of a specified prover message ($j > 0$).*

Remark 3.6. In the context of tensor IOPs, we often view a proof message $\Pi \in \mathbb{F}^{\ell \cdot k^t}$ as consisting of ℓ "sub-messages" $\pi_1, \ldots, \pi_\ell$ each in $\mathbb{F}^{k^t}$. In this case, when describing a protocol, we may make a tensor query $q_1 \otimes \cdots \otimes q_t$ to one of the sub-messages π, with the understanding that one can specify an indicator vector $q_0 \in \mathbb{F}^\ell$ so that $\langle q_0 \otimes q_1 \otimes \cdots \otimes q_t, \Pi \rangle = \langle q_1 \otimes \cdots \otimes q_t, \pi \rangle$.

4 Systematic reverse-multiplication-friendly embeddings

We recall the definition of an RMFE (from [**CascudoCXY18**]) and introduce the systematic property that we use in this work. Then in Section 4.1 and in Section 4.2 we prove that two RMFE constructions from [**CascudoCXY18**] are systematic. The first construction is based on polynomial interpolation and the second construction is based on algebraic-geometry codes. The first construction is a warmup, and the second construction is the one that we use in our IOP, via the asymptotic corollaries that we state in Section 4.3.

Definition 4.1. *Let p be a prime power and $\mathbb{F}_p$ the field with p elements. Let $a, b \geq 1$ be integers. A pair (ϕ, ψ) is a $(a, b)_p$-**reverse-multiplication-friendly embedding (RMFE)** if $\phi \colon \mathbb{F}_p^a \to \mathbb{F}_{p^b}$ and $\psi \colon \mathbb{F}_{p^b} \to \mathbb{F}_p^a$ are $\mathbb{F}_p$-linear maps such that, for every $\vec{x}, \vec{y} \in \mathbb{F}_p^a$,*

$$\vec{x} \circ \vec{y} = \psi\Big(\phi(\vec{x}) \cdot \phi(\vec{y})\Big) \ .$$

Definition 4.2. *An $(a, b)_p$-RMFE (ϕ, ψ) is* **systematic** *if there exists an $\mathbb{F}_p$-basis $\{e_j\}_{j \in [b]}$ of $\mathbb{F}_{p^b}$ such that for all $x = (x_1, \ldots, x_a) \in \mathbb{F}_p^a$, we have $\phi(x) = (x_1, \ldots, x_a, 0, \ldots, 0)$, where the i-th entry of $\phi(x)$ is the coefficient with respect to the i-th basis vector e_i.*

4.1 Systematic RMFE from polynomial interpolation

We show that the $(k, 2k-1)_p$-RMFE construction in [**CascudoCXY18**] based on polynomial interpolation is systematic. We do not use this construction in our main theorem (as the RMFE requires p to be large); however, this construction serves as a warm-up for the construction in the next section.

In the definition below we use this notation: for every positive integer t, the formal symbol ∞_t acts as an evaluation point in that, for every $f \in \mathbb{F}_p[X]_{<t}$, we define $f(\infty_t)$ to be the coefficient of X^{t-1} in f.

Definition 4.3 ([**CascudoCXY18**])**.** *Fix a choice of:*

- *positive integer k and arbitrary prime power p such that $1 \leq k \leq p+1$;*
- *an element $\zeta \in \mathbb{F}_{p^{2k-1}}$ such that $\mathbb{F}_{p^{2k-1}} = \mathbb{F}_p[\zeta]$;*
- *pairwise distinct elements $\{\alpha_1, \ldots, \alpha_k\}$ in $\mathbb{F}_p \cup \{\infty_{k+1}\}$.*

The maps are as follows.

- Encoding map: $\phi_{\mathrm{int}} \colon \mathbb{F}_p^k \to \mathbb{F}_{p^{2k-1}}$ *maps $(x_1, \ldots, x_k) \mapsto f(\zeta)$ where f is the unique polynomial in $\mathbb{F}_p[X]_{<k}$ such that $f(\alpha_i) = x_i$ for every $i \in [k]$.*

- Decoding map: $\psi_{\mathrm{int}} \colon \mathbb{F}_{p^{2k-1}} \to \mathbb{F}_p^k$ *maps $\beta \mapsto (f(\alpha'_1), \ldots, f(\alpha'_k))$ where f is the unique polynomial in $\mathbb{F}_p[X]_{<2k-1}$ such that $\beta = f(\zeta)$. Here $\alpha'_i := \alpha_i$ if $\alpha_i \in \mathbb{F}_p$ and $\alpha'_i := \infty_{2k-1}$ if $\alpha_i = \infty_{k+1}$.*

Lemma 4.4 ([**CascudoCXY18**])**.** *$(\phi_{\mathrm{int}}, \psi_{\mathrm{int}})$ in Definition 4.3 is a $(k, 2k-1)_p$-RMFE.*

Theorem 4.5. *$(\phi_{\mathrm{int}}, \psi_{\mathrm{int}})$ in Definition 4.3 is systematic.*

Proof. The interpolated polynomial for an input $\vec{x} = (x_1, \ldots, x_k) \in \mathbb{F}_p^k$ is $f(X) = \sum_{i=1}^k x_i L_i(X)$ where $L_i(X) = \prod_{j \in [k] \setminus \{i\}} \frac{X - x_j}{x_i - x_j}$ is the i-th Lagrange polynomial. By Definition 4.3, we know that $\phi_{\mathrm{int}}(\vec{x}) = f(\zeta) = \sum_{i=1}^k x_i L_i(\zeta)$ where $\zeta \in \mathbb{F}_{p^{2k-1}}$ is such that $\mathbb{F}_{p^{2k-1}} = \mathbb{F}_p[\zeta]$.

It is well-known that Lagrange polynomials are a basis; specifically, $\{L_1(X), \ldots, L_k(X)\}$ is a basis for $\mathbb{F}_p[X]_{<k}$. We prove this claim for completeness. Consider any decomposition of the zero polynomial

$0 = a_1 L_1(X) + \cdots + a_k L_k(X)$ under this Lagrange basis. By evaluating the zero polynomial at every x_i for $i \in [k]$, we get that $a_1 = \cdots = a_k = 0$. Since the zero polynomial has a unique decomposition in this basis, we conclude that $\{L_1(X), \ldots, L_k(X)\}$ are linearly independent. Thus, it forms a basis for $\mathbb{F}_p[X]_{<k}$.

We conclude by noting that $\{L_1(\zeta), \ldots, L_k(\zeta), \zeta^k, \ldots, \zeta^{(2k-1)-1}\}$ is a basis for $\mathbb{F}_{p^{2k-1}} = \mathbb{F}_p[\zeta]$ and the basis makes $(\phi_{\text{int}}, \psi_{\text{int}})$ systematic since $\phi_{\text{int}}(\vec{x}) = \sum_{i=1}^{k} x_i L_i(\zeta)$. $\square$

4.2 Systematic RMFE from algebraic geometry

We show that the $(k, m)_p$-RMFE construction in [**CascudoCXY18**] based on algebraic-geometry codes is systematic. Below, we assume familiarity with certain algebraic-geometry concepts: a function field $F/\mathbb{F}_p$; the genus g of function fields; places P associated with the function fields; the number of rational places $N_1(F)$ of a function field F; Ihara's constant $A(p)$ of $\mathbb{F}_p$; a divisor G; and the Riemann–Roch space $\mathcal{L}(G)$ associated with G. We refer the reader to the book by Stichtenoth [**Stichtenoth08**] for explanations of these notions.

Definition 4.6 ([**CascudoCXY18**])**.** *Fix a choice of:*

- *a function field $F/\mathbb{F}_p$ of genus g;*
- *k distinct rational places $P_1, \ldots, P_k$ in $F/\mathbb{F}_p$;*
- *a divisor G of F such that* $\operatorname{supp}(G) \cap \{P_1, \ldots, P_k\} = \emptyset$ *and* $\dim_{\mathbb{F}_p} \mathcal{L}(G) - \dim_{\mathbb{F}_p} \mathcal{L}(G - \sum_{i=1}^{k} P_i) = k$*;*
- *a place R of degree $m > 2 \deg G$.*

Define:

- *$\pi\colon \mathcal{L}(G) \to \mathbb{F}_p^k$ maps $f \mapsto (f(P_1), \ldots, f(P_k))$ (π is surjective since* $\ker \pi = \mathcal{L}(G - \sum_{i=1}^{k} P_i)$*);*
- *$W \subseteq \mathcal{L}(G)$ with* $\dim_{\mathbb{F}_p} W = k$ *such that π induces an isomorphism between W and $\mathbb{F}_p^k$;*
- *$\tau\colon \mathcal{L}(2G) \to \mathbb{F}_{p^m}$ maps $f \mapsto f(R) \in \mathbb{F}_{p^m}$ (τ is $\mathbb{F}_p$-linear and injective since* $m = \deg R > \deg 2G$*).*

The maps are as follows.

- Encoding map: $\phi_{\text{ag}}\colon \pi(W) = \mathbb{F}_p^k \to \mathbb{F}_{p^m}$ *maps $(x_1, \ldots, x_k) \mapsto f(R)$ where f is the unique polynomial in W such that $f(P_i) = x_i$ for every $i \in [k]$.*

- Decoding map: ψ_{ag} *to be the linear extension of ψ'_{ag} from* $\operatorname{Im}(\tau)$ *to $\mathbb{F}_{p^m}$ where $\psi'_{\text{ag}}\colon \operatorname{Im}(\tau) \to \mathbb{F}_p^k$ maps $f(R) \mapsto (f(P_1), \ldots, f(P_k))$ where $f \in \mathcal{L}(2G)$ is uniquely determined by $f(R)$.*

Lemma 4.7 ([**CascudoCXY18**])**.** *$(\phi_{\text{ag}}, \psi_{\text{ag}})$ in Definition 4.6 is a $(k, m)_p$-RMFE.*

Theorem 4.8. *$(\phi_{\text{ag}}, \psi_{\text{ag}})$ in Definition 4.6 is systematic.*

Proof. We first assume that we have a basis $\{e_j\}_{j \in [m]}$ for $\mathbb{F}_{p^m}$ such that the subset $\{e_j\}_{j \in [k]}$ of $\{e_j\}_{j \in [m]}$ (reordered if necessary) forms a basis for $\operatorname{Im}(\phi_{\text{ag}})$ and prove that this basis makes the RMFE systematic. Let $\{e'_i\}_{i \in [k]}$ be the standard basis of $\mathbb{F}_p^k$ (the i-th entry of e'_i is 1 and all other entries are 0). The map π induces an isomorphism between W and $\mathbb{F}_p^k$, which means that each $e'_i \in \mathbb{F}_p^k$ uniquely determines some $f_i \in W \subseteq \mathcal{L}(G)$. Let $y_i := f_i(R) \in \mathbb{F}_{p^m}$. We know that $y_i = \sum_{j=1}^{k} a_{ij} e_j$ because $y_i \in \operatorname{Im}(\phi_{\text{ag}})$. Note that $\{y_i\}_{i \in [k]}$ is linearly independent since ϕ_{ag} is an injective linear map, and if we replace $\{e_j\}_{j \in [k]}$ with $\{y_i\}_{i \in [k]}$, the resulting set $\{y_1, \ldots, y_k, e_{k+1}, \ldots, e_m\}$ forms a basis that makes $(\phi_{\text{ag}}, \psi_{\text{ag}})$ systematic since $\{y_i\}_{i \in [k]}$ are the image of basis vectors of the domain of ϕ_{ag}.

We are left to construct a basis for $\mathbb{F}_{p^m}$ whose first k elements form a basis for $\mathrm{Im}(\phi_{\mathrm{ag}})$. Let $\mathcal{B}$ be a basis for the Riemann–Roch space $\mathcal{L}(G - \sum_{i=1}^{k} P_i)$. Since $\mathcal{L}(G - \sum_{i=1}^{k} P_i)$ has finite dimension, $\mathcal{B}$ is a finite set. Since we have $\ker \pi = \mathcal{L}(G - \sum_{i=1}^{k} P_i)$, with the fact that ϕ_{ag} maps vectors in $\mathbb{F}_p^k$ to the evaluation of the high degree place R of a unique polynomial in $W \subseteq \mathcal{L}(G)$, we can conclude that $\mathrm{Im}(\phi_{\mathrm{ag}}) \cong \mathcal{L}(G)/\mathcal{L}(G - \sum_{i=1}^{k} P_i)$. Therefore, if we found a basis for $\mathcal{L}(G)/\mathcal{L}(G - \sum_{i=1}^{k} P_i)$, it would naturally give a basis for $\mathrm{Im}(\phi_{\mathrm{ag}})$. To do that, we start with finding a basis for $\mathcal{L}(G - \sum_{i=2}^{k} P_i)/\mathcal{L}(G - \sum_{i=1}^{k} P_i)$, then we proceed inductively to obtain a basis for $\mathcal{L}(G)/\mathcal{L}(G - \sum_{i=1}^{k} P_i)$. Let $k(P_1)$ be the residue field of P_1. Consider the linear map $\chi\colon \mathcal{L}(G - \sum_{i=2}^{k} P_i)/\mathcal{L}(G - \sum_{i=1}^{k} P_i) \to k(P_1)$ defined as $\chi(f) = f \bmod P_1$. Since $\mathcal{L}(G - \sum_{i=2}^{k} P_i)/\mathcal{L}(G - \sum_{i=1}^{k} P_i)$ is of dimension 1, any f not in $\ker(\chi)$ would be a basis for $\mathcal{L}(G - \sum_{i=2}^{k} P_i)/\mathcal{L}(G - \sum_{i=1}^{k} P_i)$. □

Remark 4.9. Theorem 4.5 is a special case of Theorem 4.8. We started with the canonical basis $\{1, \zeta, \ldots, \zeta^{(2k-1)-1}\}$ of $\mathbb{F}_{p^{2k-1}}$ and replace the first k elements by the evaluations of degree k Lagrange polynomials at ζ, which is a basis for $\mathbb{F}_p^k$ and $\mathrm{Im}(\phi_{\mathrm{int}}) \subseteq \mathbb{F}_p^k$.

4.3 Asymptotic results

We need a family of systematic $(a, b)_p$-RMFE such that $b \in O(a)$. Below we explain how such a family exists for every finite field $\mathbb{F}_p$.

Proposition 4.10 ([**Stichtenoth08**]). *For every function field $F/\mathbb{F}_p$ and $b \in \mathbb{N}$ with $2g+1 \leq p^{(b-1)/2}(\sqrt{p}-1)$, there exists a place in F of degree b. In particular, this holds for every $m \geq 4g+3$, regardless of p.*

Remark 4.11. For $a \geq 2$, any $b \geq 2a + 4g - 1$ satisfies the inequality in Proposition 4.10. Therefore, as long as $F/\mathbb{F}_p$ has at least a distinct rational places, we can construct $(a, b)_p$-RMFE for any $b \geq 2a + 4g - 1$ according to Definition 4.6.

Theorem 4.12 ([**CascudoCXY18**]). *For every prime power p, there exists a family of $(a, b)_p$-RMFE (via the construction in Definition 4.6) where $\lim_{a\to\infty} \frac{b}{a} = 2 + \frac{4}{A(p)}$ (here $A(p)$ is Ihara's constant).*

Proof. Consider a family $\{F_l\}_l$ of function fields over $\mathbb{F}_p$ of growing genus $g_l \to \infty$ such that $N_1(F_l)/g_l \to A(p)$. Let $a = N_1(F_l)$ and $b = 2a + 4g_l - 1$. Note that $N_1(F_l)$ is the number of distinct rational places of F_l, let $\{P_1, \ldots, P_k\}$ be these places. We take G to be a degree-$(a + 2g_l - 1)$ divisor such that $\mathrm{supp}(G) \cap \{P_1, \ldots, P_k\} = \emptyset$. Since $\deg G \geq 2g_l - 1$ and $\deg G - \sum_{i=1}^{k} P_i \geq 2g_l - 1$, the Riemann Theorem guarantees that $\dim_{\mathbb{F}_p} \mathcal{L}(G) - \dim_{\mathbb{F}_p} \mathcal{L}(G - \sum_{i=1}^{k} P_i) = k$. Then by Lemma 4.7, the construction in Definition 4.6 yields a $(a, b)_p$-RMFE. □

Remark 4.13. For $p = 2$, [**DuursmaM2011**] proves the lower bound $A(2) \geq 0.31$. So by Theorem 4.12 there exists a family of $(a, b)_2$-RMFE with $\lim_{a\to\infty} \frac{b}{a} \leq 2 + \frac{4}{0.31} \approx 14.903$. The lower bound is constructive: [**DuursmaM2011**] gives a construction for an infinite sequence of function fields $\{F_l\}_l$ over $\mathbb{F}_2$ such that $N_1(F_l)/g_l \to 0.31$.

[**CascudoCXY18**] also gives an explicit family of $(a, b)_2$-RMFE with $\lim_{a\to\infty} \frac{b}{a} \to c \approx 4.92$. Instead of finding a bound on $A(2)$ and then applying Theorem 4.12, they build a family of $(a, b)_{32}$-RMFE where $\lim_{a\to\infty} \frac{b}{a} = \frac{62}{21}$, and then show that the concatenation of a $(3, 5)_2$-RMFE and the family of $(a, b)_{32}$-RMFE is a family of $(3a, 5b)_2$-RMFE where $\lim_{a\to\infty} \frac{5b}{3a} \approx 4.92$.

5 Biased generators

We define two biased generators that are later used in our constructions.

Definition 5.1. *Let $G\colon \mathbb{F}^s \to \mathbb{F}^n$ be a function. We say that G is an* ϵ**-biased generator** *if*

$$\max_{w \in \mathbb{F}^n \setminus \{0\}} \Pr_{x \in \mathbb{F}^s}[\langle w, G(x)\rangle = 0] \le \epsilon \ .$$

Lemma 5.2 (**[BootleCL22]**)**.** *Let $G\colon \mathbb{F}^s \to \mathbb{F}^n$ be an ϵ-biased generator and let $G'\colon \mathbb{F}^s \to \mathbb{F}^n$ be an ϵ'-biased generator. Then, the function $G \otimes G'\colon (x, x') \mapsto G(x) \otimes G'(x')$ is an $(\epsilon + \epsilon')$-biased generator.*

Definition 5.3. *Let $G_{\mathsf{sub}} \colon \mathbb{F}_{p^\lambda}^s \to \mathbb{F}_p^{\lambda \times \lambda b'}$ be a function. We say that G_{sub} is a* (p, λ)**-subspace** ϵ_{sub}**-biased generator** *if for any $\mathbb{F}_p$-linear subspace $H \subseteq \mathbb{F}_{p^\lambda}$,*

$$\max_{v \in \mathbb{F}^{\lambda b'} \setminus H^{\lambda b'}} \Pr_{r \leftarrow \mathbb{F}_{p^\lambda}^s} \left[G_{\mathsf{sub}}(r) v = \vec{0} \mod H^{\lambda b'} \right] \le \epsilon_{\mathsf{sub}} \ .$$

We note that (p, λ)-subspace ϵ_{sub}-biased generators can be constructed from ϵ-biased generators. A proof is given in Section 5.1.

5.1 A subspace biased generator from an ϵ-biased generator

In this section we provide a construction for the (p, λ)-subspace ϵ-biased generator for $\mathbb{F}$ which is used in the modular lincheck protocol. The construction is done by doing transformation to an ϵ-biased generator over $\mathbb{F}_{p^\lambda}$.

Before introducing the construction we make two observations about fields of the form $\mathbb{F}_{p^\lambda}$.

Observation 5.4. *For an degree-λ irreducible polynomial P in $\mathbb{F}_p[X]$, $\mathbb{F}_{p^\lambda} \cong \mathbb{F}_p[X]/(P)$. Every element $f \in \mathbb{F}_p[X]/(P)$ has a matrix representation $M_f \in \mathbb{F}_p^{\lambda \times \lambda}$ such that for any $g = \sum_{i=0}^{\lambda-1} a_i X^i \in \mathbb{F}_p[X]/(P)$ with associated coefficient vector $\vec{g}$, $M_f \cdot \vec{g}$ is the coeffcient vector for $f \cdot g \in \mathbb{F}_p[X]/(P)$.*

Construction 5.5. Given an ϵ-biased generator $G\colon \mathbb{F}_{p^\lambda}^s \to \mathbb{F}_{p^\lambda}^{b'}$ as input. Fix $r \in \mathbb{F}_{p^\lambda}^s$. Use $G(r)_i \in \mathbb{F}_{p^\lambda}$ to denote the i-th element of $G(r)$, and $G_{\mathsf{sub}}(r)_i \in \mathbb{F}_p^{\lambda \times \lambda}$ to denote the i-th $\lambda \times \lambda$ block of $G_{\mathsf{sub}}(r)$. Then $G_{\mathsf{sub}}(r)_i$ is defined to be $M_{G(r)_i}$, the matrix representation of $G(r)_i$ from Observation 5.4.

Lemma 5.6. *Given an ϵ-biased generator $G\colon \mathbb{F}_{p^\lambda}^s \to \mathbb{F}_{p^\lambda}^{b'}$, the construction in Construction 5.5 gives a (p, λ)-subspace ϵ-biased generator G_{sub} for $\mathbb{F}$.*

Proof. By definition of an ϵ-biased generator, we know that for any $z \in \mathbb{F}_p^{b'} \setminus \{\vec{0}\}$,

$$\Pr_{r \leftarrow \mathbb{F}_{p^\lambda}^s} [\langle G(r), z\rangle = 0] \le \epsilon \ . \tag{2}$$

We now show that G_{sub} in Construction 5.5 is indeed a (p, λ)-subspace ϵ-biased generator for $\mathbb{F}$. Let $\theta\colon \mathbb{F}_p^\lambda \to \mathbb{F}_{p^\lambda}$ be the isomorphism that maps a vector $b \in \mathbb{F}_p^\lambda$ to $\sum_{j=0}^{\lambda-1} b_j X^j \in \mathbb{F}_{p^\lambda}$. Then we note that for any $x \in \mathbb{F}_{p^\lambda}$ and $i = 1, \ldots, \lambda$,

$$G(r)_i \cdot \theta(b) = \theta\left(M_{G(r)_i} \cdot b\right) = \theta\left(G_{\mathsf{sub}}(r)_i \cdot b\right)$$

For any $H \subseteq \mathbb{F}$ a $\mathbb{F}_p$-linear subspace of $\mathbb{F}$, let $j = \dim H$. We can find a base $e_1, \dots, e_k$ of $\mathbb{F}$ such that $e_1, \dots, e_j$ is a basis for H. Consider any $v \in \mathbb{F}^{\lambda b'} \setminus H^{\lambda b'}$. Then represent v using this basis as

$$v = \sum_{i=1}^{k} e_i c_i$$

where $c_i \in \mathbb{F}_p^{\lambda b'}$.

We have that

$$\begin{aligned} G_{\mathsf{sub}}(r)v &= \sum_{i=1}^{k} e_i G_{\mathsf{sub}}(r) c_i \\ &= \sum_{i=1}^{k} e_i \theta^{-1}(\langle G(r), \theta(c_i)\rangle) \end{aligned}$$

Suppose $G_{\mathsf{sub}}(r)v = \vec{0} \mod H^{\lambda}$. Then for all $i > j$ we have $\theta^{-1}(\langle G(r), \theta(c_i)\rangle) = \vec{0}$. Additionally, since $v \notin H^{\lambda b'}$, there exists some $i^* > j$ such that $c_{i^*} \neq \vec{0}$. Therefore

$$\begin{aligned} \Pr_{r \leftarrow \mathbb{F}^s_{p^\lambda}} \left[G_{\mathsf{sub}}(r)v = \vec{0} \mod H^{\lambda} \right] &\leq \Pr_{r \leftarrow \mathbb{F}^s_{p^\lambda}} \left[\theta^{-1}(\langle G(r), \theta(c_{i^*})\rangle) = \vec{0} \right] \\ &= \Pr_{r \leftarrow \mathbb{F}^s_{p^\lambda}} \left[\langle G(r), \theta(c_{i^*})\rangle = 0 \right] \leq \epsilon \end{aligned}$$

The last inequality holds by definition of G. Thereby we complete the proof. $\square$

Remark 5.7. Note that for our construction to work, a weaker ϵ-biased generator G suffices. We only requires that Equation (2) holds for all $z \in \mathbb{F}_p^{b'} \setminus \{\vec{0}\}$ instead of for all $z \in \mathbb{F}_{p^\lambda}^{b'} \setminus \{\vec{0}\}$.

We also note that computing $G_{\mathsf{sub}}(r)$ takes $O(\mathsf{o}_G)$ $\mathbb{F}_{p^\lambda}$ operations, where o_G denotes the number of operations needed to compute $G(r)$.

5.2 A biased generator computable in linear time

We describe a biased generator that is computable in linear time, over any given field $\mathbb{F}$. The construction is a generalization of the biased generator in [**RonZewiR22**] for the boolean field $\mathbb{F}_2$. First we describe a construction based on any linear error correcting code over $\mathbb{F}$, and then we deduce the desired result via the existence of linear codes of constant rate and relative distance that are efficiently encodable.

Lemma 5.8. *Let $\mathcal{C} \subseteq \mathbb{F}^n$ be a linear code with rate r and relative distance δ. If $\mathcal{C}$'s generator matrix can be computed in time $O(n^c)$, then for any integer t there exists a $(\frac{|\mathbb{F}|-1}{|\mathbb{F}|}(1-\delta^c)^t + \frac{1}{|\mathbb{F}|})$-biased generator $G \colon \mathbb{F}^t \times [n]^{tc} \to \mathbb{F}^{(rn)^c}$ that is computable with $O(tcn^c)$ $\mathbb{F}$-operations.*

Proof. Let $M \in \mathbb{F}^{n \times rn}$ be the generating matrix of the code $\mathcal{C}$, and M_i be its i-th row. Then consider the function $S(i) = M_i$. We observe that first S can be computed in time $O(n^c)$, and second $S \colon [n] \to \mathbb{F}^{rn}$ is a $(1-\delta)$-biased generator. This is because for every $\vec{y} \in \mathbb{F}^{rn} \setminus \{\vec{0}\}$ it holds that

$$\Pr_{i \leftarrow [n]}[\langle S(i), \vec{y}\rangle = 0] = \Pr_{i \leftarrow [n]}[\langle M_i, \vec{y}\rangle = 0] = 1 - \|M\vec{y}\|_0 \leq 1 - \delta \ .$$

We prove that the tensor function $S^{\otimes i}\colon i[n] \to \mathbb{F}^{(rn)^i}$ is a $(1-\delta^i)$-biased generator over $\mathbb{F}$, b induction on i.

Base case. For $i = 1$, the result comes trivially.

Inductive step. Suppose this fact holds for $i-1$. Consider any $Y \in \mathbb{F}^{(rn)^{i-1}\times rn}$ that is not all zero. Use $\vec{Y}$ to denote the vectorization of Y, which stacks the columns of Y to make it a column vector. Then we have

$$\begin{aligned}
&\Pr_{(x,x')\leftarrow[n]^{i-1}\times[n]}[\langle S^{\otimes i}(x)\otimes S(x'), \vec{Y}\rangle = 0]\\
&= \Pr_{(x,x')\leftarrow[n]^{i-1}\times[n]}[S^{\otimes i}(x)^T Y S(x') = 0]\\
&= \Pr_{x'\leftarrow[n]}[YS(x') = \vec{0}] + \Pr_{x'\leftarrow[n]}[YS(x') \neq \vec{0}] \cdot \Pr_{x\leftarrow[n]^{i-1}}[\langle S^{\otimes i}(x), YS(x')\rangle = 0 \mid YS(x') \neq \vec{0}]\\
&\leq (1-\delta) + \delta\cdot(1-\delta^{i-1})\\
&= 1-\delta^i \ .
\end{aligned}$$

Therefore the function $S^{\otimes c}[n]^c \to \mathbb{F}^{(rn)^c}$ is a $(1-\delta^c)$-biased generator.

We use $S^{\otimes c}$ to construct the ϵ-biased generator $G\colon \mathbb{F}^t \times [n]^{tc} \to \mathbb{F}^{(rn)^c}$ with $\epsilon = \frac{|\mathbb{F}|-1}{|\mathbb{F}|}(1-\delta^c)^t + \frac{1}{|\mathbb{F}|}$. For any $z \in \mathbb{F}^t$ and $x = (x_i)_{i\in[t]}$ where $x_i \in [k]^c$, we define $G(z,x) := \sum_{i=1}^t z_i S^{\otimes c}(x_i)$. For every $\vec{y} \in \mathbb{F}^{(rn)^c} \setminus \{\vec{0}\}$,

$$\begin{aligned}
&\Pr_{z\leftarrow\mathbb{F}^t, x\leftarrow[n]^{tc}}[\langle G(z,x), \vec{y}\rangle = 0]\\
&= \Pr_{z\leftarrow\mathbb{F}^t, x\leftarrow[n]^{tc}}[\sum_{i=1}^t z_i\langle S^{\otimes c}(x_i), \vec{y}\rangle = 0]\\
&= \Pr_{x\leftarrow[n]^{tc}}[\forall\, i, \langle S^{\otimes c}(x_i), \vec{y}\rangle = 0] + \Pr_{x\leftarrow[n]^{tc}}[\exists\, i, \langle S^{\otimes c}(x_i), \vec{y}\rangle \neq 0] \cdot \max_{\vec{y}'\in\mathbb{F}^t\setminus\{\vec{0}\}} \Pr_{z\sim\mathbb{F}^t}[\langle z, \vec{y}'\rangle = 0]\\
&\leq (1-\delta^c)^t + (1-(1-\delta^c)^t)\cdot\frac{1}{|\mathbb{F}|} = \frac{|\mathbb{F}|-1}{|\mathbb{F}|}(1-\delta^c)^t + \frac{1}{|\mathbb{F}|} \ .
\end{aligned}$$

To compute G, we call S tc times, compute t order c tensor products, and calculate the linear combination of t vectors in $\mathbb{F}^{(rn)^c}$. This in total takes $O(tcn^c + t(rn)^c) = O(tcn^c)$ operations in $\mathbb{F}$. $\square$

Lemma 5.9 ([DrukI14]). *There exist constants $d_1 > 1$ and $d_2 > 0$ such that for every finite field $\mathbb{F}$ there exists a family of linear codes with code length $\lfloor d_1 k\rfloor$, dimension k, and minimum distance $\lceil d_2 k\rceil$ over $\mathbb{F}$ which can be encoded by a uniform family of linear-size arithmetic circuits consisting of addition and fan-out gates only.*

Corollary 5.10. *For every finite field $\mathbb{F}$, and constant $\epsilon > \frac{1}{|\mathbb{F}|}$, there exists a family of ϵ-biased generators $G\colon \mathbb{F}^{\log_{|\mathbb{F}|}(n)} \to \mathbb{F}^{O(n)}$ that can be computed with $O(n)$ $\mathbb{F}$-operations.*

Proof. Invoke Lemma 5.8 with the codes in Lemma 5.9. $\square$

6 Tensor IOP for R1CS over every field

We construct a tensor-query IOP for R1CS over fields $\mathbb{F}_p$ for any prime power p.

Definition 6.1 (R1CS)**.** *The indexed relation R_{R1CS} is the set of all triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}_p, n_{\mathrm{row}}, n_{\mathrm{col}}, A, B, C), (n_{\mathrm{in}}, x), w\big)$$

where $\mathbb{F}_p$ is a finite field, A, B, C are matrices in $\mathbb{F}_p^{n_{\mathrm{row}} \times n_{\mathrm{col}}}$ each with at most M non-zero entries, $x \in \mathbb{F}_p^{n_{\mathrm{in}}}$, $w \in \mathbb{F}_p^{n_{\mathrm{col}} - n_{\mathrm{in}}}$, and $z := (x, w) \in \mathbb{F}_p^{n_{\mathrm{col}}}$ is a vector such that $Az \circ Bz = Cz$. (Here "$\circ$" is the entry-wise product.)

Theorem 6.2. *For all prime powers p, and every finite field $\mathbb{F}_{p^b}$ such that $p^b \in \Omega(\log n_{\mathrm{col}})$, given a systematic $(a, b)_p$-RMFE (ϕ, ψ) such that $a \in \Theta(b)$, there is a $(\mathbb{F}_{p^b}, k, t)$-tensor IOP, with non-adaptive queries, for the indexed relation R_{R1CS} that supports instances over $\mathbb{F}_p$ with $n_{\mathrm{row}} = n_{\mathrm{col}} = a \cdot k^t$, $n_{\mathrm{in}} = a \cdot \ell_{\mathrm{in}} \cdot k^{t_{\mathrm{in}}}$ and has the following parameters:*
- *soundness error is $O(1)$;*
- *round complexity is $O(\log(n_{\mathrm{row}}/a))$;*
- *proof length is $O(n_{\mathrm{row}}/a)$ elements in $\mathbb{F}_{p^b}$;*
- *query complexity is $O(1)$;*
- *the prover sends $O(\log(n_{\mathrm{row}}/a))$ non-oracle messages over $\mathbb{F}_{p^b}$;*
- *the prover uses $O(M + n_{\mathrm{row}})$ $\mathbb{F}_p$-operations and $O(tn_{\mathrm{row}}/a)$ $\mathbb{F}_{p^b}$-operations;*
- *the verifier uses $O(M + n_{\mathrm{row}})$ $\mathbb{F}_p$-operations and $O(tk)$ $\mathbb{F}_{p^b}$-operations;*
- *the verifier has randomness complexity $O(\log(n_{\mathrm{row}}/a))$ over $\mathbb{F}_{p^b}$ and $O(\log(n_{\mathrm{row}}/a))$ over $\mathbb{F}_p$.*

Here M denotes the number of non-zero entries in the instance matrices A, B, C.

[where do we say that Theorem 6.2 is a corollary of Theorem 6.11 using certain biased generators? — *Jonathan***]** ★

Our construction involves first embedding the instance matrices into a large field $\mathbb{F}_{p^b}$ using a systematic RMFE, and then reducing the R1CS relation over $\mathbb{F}_p$ to a different relation over $\mathbb{F}_{p^b}$. This relation requires the embedded instance matrices to satisfy a set of modular-linear equalities. So we need to construct a protocol for the modular-linear relation. In its constructon, we make use of a tensor-query protocol for the multi-lincheck relation. This section is organized as follows. We first introduce the protocol for the multi-lincheck relation. Then based on that, we construct the modular-linear relation protocol. Building on top of that, we provide a tensor IOP for R_{R1CS} that achieves linear time.

6.1 Multi-lincheck

We describe a tensor-query IOP for the multi-lincheck relation, which tests the sum of several matrix products. We first define the twisted scalar product relation, which is used as a subroutine in the multi-lincheck protocol.

Definition 6.3. *The* **twisted scalar product** *relation R_{TSP} is the set of triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\perp, (\mathbb{F}, n, \vec{y}, \tau), (\vec{a}, \vec{b}))$$

where $\vec{a}, \vec{b}, \vec{y} \in \mathbb{F}^n$, $\tau \in \mathbb{F}$, and $\langle \vec{a} \circ \vec{y}, \vec{b} \rangle = \tau$.

Lemma 6.4 ([**BootleCG20**]). *For every finite field $\mathbb{F}$ and positive integers k, t, there is a $(\mathbb{F}, k, t)$-tensor exact IOPP for the indexed relation R_{TSP} that supports instances over $\mathbb{F}$ with $n = \ell \cdot k^t$ and $\vec{y} = \vec{y}_0 \otimes \vec{y}_1 \otimes \cdots \otimes \vec{y}_t$ for $\vec{y}_0 \in \mathbb{F}^\ell$, $\vec{y}_1, \ldots, \vec{y}_t \in \mathbb{F}^k$, and has the following parameters: soundness error is $O(\log n / |\mathbb{F}|)$; round complexity is $O(\log n)$; query complexity is $O(1)$; the prover sends $O(\log n)$ non-oracle messages in $\mathbb{F}$ and the proof length is $O(n)$ elements in $\mathbb{F}$; the prover uses $O(n)$ field operations; the verifier uses $O(\ell + tk)$ field operations; and the verifier has randomness complexity $O(\log n)$ elements in $\mathbb{F}$.*

Definition 6.5. *The* **multi-lincheck** *relation R_{mlin} is the set of triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (((U_i)_{i\in[h]}, \vec{b}), (\mathbb{F}, n_{\mathrm{row}}, n_{\mathrm{col}}, h, (M_i)_{i\in[h]}), (\vec{x}_i)_{i\in[h]})$$

where $U_i \in \mathbb{F}^{n_{\mathrm{row}} \times n_{\mathrm{col}}}$ has M_i non-zero entries, $\vec{x}_i \in \mathbb{F}^{n_{\mathrm{col}}}$, $\vec{b} \in \mathbb{F}^{n_{\mathrm{row}}}$, and $\sum_{i=1}^{h} U_i \vec{x}_i = \vec{b}$.

Theorem 6.6. *For every finite field $\mathbb{F}$ and positive integers k, t, there is a $(\mathbb{F}, k, t)$-tensor IOP, with non-adaptive queries, for the indexed relation R_{mlin} that supports instances over $\mathbb{F}$ with $n_{\mathrm{col}} = \ell \cdot k^t$ and arbitrary n_{row} and has the following parameters:*

- *perfect completeness;*
- *soundness error is $O(\log n_{\mathrm{col}} / |\mathbb{F}|)$;*
- *round complexity is $O(\log n_{\mathrm{col}})$;*
- *proof length is $O(hn_{\mathrm{col}})$ elements in $\mathbb{F}$;*
- *query complexity is $O(h)$;*
- *the prover sends $O(h(n_{\mathrm{row}} + \log n_{\mathrm{col}}))$ non-oracle messages;*
- *the prover uses $O(hn_{\mathrm{col}} + hn_{\mathrm{row}} + \sum_{i=1}^{h} M_i)$ field operations;*
- *the verifier uses $O(\sum_{i=1}^{h} M_i + h \cdot (\ell + kt))$ field operations;*
- *the verifier has randomness complexity $O(h \log n_{\mathrm{col}})$ elements in $\mathbb{F}$.*

Construction 6.7 (tensor IOP for R_{mlin})**.** The prover $\mathbf{P}$ takes as input the index $\mathbb{i} = ((U_i)_{i\in[h]}, \vec{b})$, instance $\mathbb{x} = (\mathbb{F}, n_{\mathrm{row}}, n_{\mathrm{col}}, h, (M_i)_{i\in[h]})$, and witness $\mathbb{w} = (\vec{x}_i)_{i\in[h]}$, while the verifier $\mathbf{V}$ takes as input $\mathbb{i}$ and $\mathbb{x}$.

- The prover $\mathbf{P}$ computes $\vec{b}_i = U_i \vec{x}_i \in \mathbb{F}^{n_{\mathrm{row}}}$. The prover $\mathbf{P}$ sends the oracle messages $\mathbb{w}$ and non-oracle messages $(\vec{b}_i)_{i\in[h]}$.
- The verifier $\mathbf{V}$ sends uniformly random challenge vector $\vec{r} \in \mathbb{F}^{n_{\mathrm{row}}}$.
- The prover $\mathbf{P}$ computes $\tau_i = \langle \vec{r}, \vec{b}_i \rangle \in \mathbb{F}$ and $\vec{a}_i = \vec{r}^{\intercal} U_i \in \mathbb{F}^{n_{\mathrm{col}}}$. The prover $\mathbf{P}$ sends the non-oracle messages $(\tau_i)_{i\in[h]} \in \mathbb{F}^h$.
- The verifier $\mathbf{V}$ computes $\vec{a}_i = \vec{r}^{\intercal} U_i \in \mathbb{F}^{n_{\mathrm{col}}}$.
- The prover $\mathbf{P}$ and the verifier $\mathbf{V}$ engage in h TSP protocols, each of which with $\mathbb{i} = \perp$, $\mathbb{x} = (\mathbb{F}, n_{\mathrm{col}}, \vec{1}, \tau_i)$, and $\mathbb{w} = (\vec{a}_i, \vec{x}_i)$ to check that $\langle \vec{a}_i \circ \vec{1}, \vec{x}_i \rangle = \tau_i$.
- The verifier $\mathbf{V}$ computes $\langle \vec{r}, \vec{b}_i \rangle$ and checks that $\tau_i = \langle \vec{r}, \vec{b}_i \rangle$. The verifier computes $\langle \vec{r}, \vec{b} \rangle$ and checks that $\sum_{i=1}^{h} \tau_i = \langle \vec{r}, \vec{b} \rangle$.

Proof. We analyze the error and efficiency parameters of the above construction.

Completeness. Suppose $\sum_{i=1}^{h} U_i \vec{x}_i = \vec{b}$. The honest prover sends the correct values of $(\vec{b}_i)_{i\in[h]}$ and $(\tau_i)_{i\in[h]}$. Then by the completeness of the TSP protocol, each check of $\langle \vec{a}_i, \vec{x}_i \rangle = \tau_i$ passes, and by definition, the check for $\sum_{i=1}^{h} \tau_i = \langle \vec{r}, \vec{b} \rangle$ also passes. Therefore, the construction is perfectly complete.

Soundness. The soundness error is $O(\log n_{\mathrm{col}} / |\mathbb{F}|)$: Suppose $\sum_{i=1}^{h} U_i \vec{x}_i \neq \vec{b}$. If the verifier accepts, then there are two possible cases:

- The honestly computed $(\tau_i)_{i\in[h]}$ satisfies that $\sum_{i=1}^h \tau_i = \langle \vec{r}, \vec{b} \rangle$: Consider $\sum_{i=1}^h \tau_i = \langle \vec{r}, \vec{b} \rangle$ as a polynomial with respect to the $n_{\rm row}$ coordinate of $\vec{r}$, each uniformly randomly sampled from $\mathbb{F}$, by Schwartz–Zippel Lemma, we have that $\Pr[\sum_{i=1}^h \tau_i = \langle \vec{r}, \vec{b} \rangle] \le 1/|\mathbb{F}| < \log n_{\rm col}/|\mathbb{F}|$.
- At least one value τ_{i^*} is not honestly computed: the probability that the verifier accepts is at most the soundness error of the TSP protocol, which is $\log n_{\rm col}/|\mathbb{F}|$.

Efficiency parameters.

- round complexity is $O(\log n_{\rm col})$: The round complexity of the protocol is dominated by the round complexity of the TSP protocol, which is $O(\log n_{\rm col})$.
- proof length is $O(hn_{\rm col})$ elements in $\mathbb{F}$: The proof length is the sum of length of $\vec{x}_i$'s and the proof length for the TSP protocols. We know that the proof length for one TSP protocol is $O(n_{\rm col})$ elements in $\mathbb{F}$, the total proof length is $O(hn_{\rm col})$.
- query complexity is $O(h)$: The query complexity is dominated by the TSP protocols, which is $O(h)$ in total.
- the prover sends $O(h(n_{\rm row}+\log n_{\rm col}))$ non-oracle messages: The non-oracle messages sent by the prover are τ_i's and $\vec{b}_i$'s, and the non-oracle messages in the TSP protocols, which in total is $O(hn_{\rm row}) + O(h\log n_{\rm col})$.
- the prover uses $O(hn_{\rm col} + hn_{\rm row} + \sum_{i=1}^h M_i)$ field operations: The prover uses $O(M_i)$ operations in $\mathbb{F}$ to compute each $\vec{b}_i$'s. Then the prover uses $O(hn_{\rm row} + \sum_{i=1}^h M_i)$ to compute τ_i's and $\vec{a}_i$'s. Then it also uses $O(h\cdot n_{\rm col})$ to finish the SP protocols. The total number of field operations is thus $O(hn_{\rm col} + hn_{\rm row} + \sum_{i=1}^h M_i)$.
- the verifier uses $O(\sum_{i=1}^h M_i + h\cdot(\ell + kt))$ field operations: The verifier uses $O(\sum_{i=1}^h M_i)$ field operations to compute $\vec{a}_i$'s. The total cost needs to account for the cost of the TSP protocols as well.
- the verifier has randomness complexity $O(h\log n_{\rm col})$ elements in $\mathbb{F}$: The randomness complexity is dominated by the randomness complexity of the TSP protocol.

□

6.2 Modular multi-lincheck protocol

Definition 6.8. *The modular multi-lincheck relation $R_{\rm Mlin_h}$ is the set of triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ((H, (U_i)_{i\in[h]}), (\mathbb{F}, \mathsf{E}, n_{\rm row}, n_{\rm col}, h), (\vec{x}_i)_{i\in[h]})$$

where $H \subseteq \mathbb{F}$ is a $\mathbb{F}_p$-linear subspace of $\mathbb{F}$, $U_i \in \mathbb{F}^{n_{\rm row}\times n_{\rm col}}$, E is a basis for the $\mathbb{F}_p$ linear space $\mathbb{F}$, and $\vec{x}_i \in \mathbb{F}^{n_{\rm col}}$ such that $\sum_{i=1}^h U_i\vec{x}_i = \vec{0} \mod H^{n_{\rm row}}$. Additionally, U_i and $\vec{x}_i$ are given in the basis E, so they can be written as $U_i = \sum_{j\in[|\mathsf{E}|]} A_{ij}e_j$ and $\vec{x}_i = \sum_{j\in[|\mathsf{E}|]} u_{ij}e_j$ where $A_{ij} \in \mathbb{F}_p^{n_{\rm row}\times n_{\rm col}}$ and $u_{ij} \in \mathbb{F}_p^{n_{\rm col}}$.

We construct a tensor IOP for the relation $R_{\rm Mlin_h}$ using a tensor IOP for $R_{\rm mlin}$ and a (p,λ)-subspace $\epsilon_{\sf sub}$-biased generator for $\mathbb{F}$. Let $\epsilon_{\rm LC}$ denote the soundness error of the multi-lincheck protocol, $\mathsf{r}_{\rm LC}$ its round complexity, $\mathsf{l}_{\rm LC}$ its proof length, $\mathsf{qc}_{\rm LC}$ its query complexity, $\mathsf{c}_{\rm LC}$ its communication complexity, $\mathsf{tp}_{\rm LC}$ its prover arithmetic complexity, $\mathsf{tv}_{\rm LC}$ its verifier arithmetic complexity, and $\mathsf{rd}_{\rm LC}$ its randomness complexity. We also use $\mathsf{o}_{G_{\sf sub}}$ to denote the arithmetic complexity of computing $G_{\sf sub}$.

Theorem 6.9. *For every prime power p, and every finite field $\mathbb{F}_{p^b}$, every $\mathbb{F}_p$-linear subspace $H \subseteq \mathbb{F}_{p^b}$, and positive integers k, t, given a (p, λ)-subspace ϵ_{sub}-biased generator $G_{\mathsf{sub}} \colon \mathbb{F}_{p^\lambda}^s \to \mathbb{F}_p^{\lambda \times n_{\text{row}}}$, there is a $(\mathbb{F}_{p^b}, k, t)$-tensor IOP, with non-adaptive queries, for the indexed relation R_{Mlin_h} that supports instances over $\mathbb{F}_{p^b}$ with $n_{\text{col}} = \ell \cdot k^t$, and has the following parameters:*
- *soundness error is $\max(\epsilon_{\mathsf{sub}}, \epsilon_{\text{LC}})$;*
- *round complexity is $O(\mathsf{r}_{\text{LC}})$;*
- *proof length is $O(h \cdot n_{\text{col}} + \mathsf{l}_{\text{LC}})$ elements in $\mathbb{F}_{p^b}$;*
- *query complexity is $O(\mathsf{qc}_{\text{LC}})$;*
- *the prover sends $O(\lambda + \mathsf{c}_{\text{LC}})$ non-oracle messages;*
- *the prover uses $O(\lambda \cdot M_{\mathsf{E}} + \mathsf{o}_{G_{\mathsf{sub}}})$ $\mathbb{F}_p$ operations and $O(h \cdot \lambda n_{\text{col}} + \mathsf{tp}_{\text{LC}})$ $\mathbb{F}_{p^b}$ operations;*
- *the verifier uses $O(\lambda \cdot (M_{\mathsf{E}} + b^3) + \mathsf{o}_{G_{\mathsf{sub}}})$ $\mathbb{F}_p$-operations and $O(\mathsf{tv}_{\text{LC}})$ $\mathbb{F}_{p^b}$-operations;*
- *the verifier has randomness complexity $O(s)$ over $\mathbb{F}_{p^\lambda}$ and $O(\mathsf{rd}_{\text{LC}})$ elements in $\mathbb{F}_{p^b}$.*

Here M_{E} denotes the number of non-zero entries in U_is' coefficient matrices.

Construction 6.10 (tensor IOP for R_{Mlin_h})**.** We construct an interactive oracle proof $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation R_{Mlin_h}.

The prover $\mathbf{P}$ takes as input the index $\mathbb{i} = (H, (U_i)_{i \in [h]})$, instance $\mathbb{x} = (\mathbb{F}, \mathsf{E}, n_{\text{row}}, n_{\text{col}}, h)$, and witness $\mathbb{w} = (\vec{x}_i)_{i \in [h]}$, while the verifier takes as input $\mathbb{i}$ and $\mathbb{x}$.

- The prover $\mathbf{P}$ sends the oracle message $\mathbb{w}$.
- The verifier $\mathbf{V}$ sends uniformly random seeds $r \in \mathbb{F}_{p^\lambda}^s$.
- The prover computes the matrix $R = G_{\mathsf{sub}}(r) \in \mathbb{F}_p^{\lambda \times n_{\text{row}}}$ and the vector $\vec{b} = \sum_{i=1}^h RU_i\vec{x}_i \in \mathbb{F}^\lambda$, and sends the non-oracle message $\vec{b}$.
- The verifier first checks that $\vec{b} = \vec{0} \mod H^\lambda$, and rejects if the check fails.
- The prover and the verifier $\mathbf{V}$ compute the matrices $B_i = RU_i \in \mathbb{F}^{\lambda \times n_{\text{col}}}$ and M_i the number of non-zero elements in B_i. Then they engage in a multi-lincheck protocol, with $\mathbb{i} = \Big((B_i)_{i \in [h]}, \vec{b}\Big)$, $\mathbb{x} = (\mathbb{F}, \lambda, n_{\text{col}}, h, (M_i)_{i \in [h]})$, and $\mathbb{w} = (\vec{x}_i)_{i \in [h]}$.

Proof. **Completeness.** Suppose the triple is in the relation and the prover is honest. Then $\sum_{i=1}^h U_i\vec{x}_i = \vec{0} \mod H^{n_{\text{row}}}$, and the prover sends the correct value $\vec{b}$. By definition $\vec{b} = \vec{0} \mod H$, and $\sum_{i=1}^h RU_i\vec{x}_i = \vec{b}$. So the construction is perfectly complete.

Soundness error. Suppose $\sum_{i=1}^h M_i\vec{x}_i \neq \vec{0} \mod H^{n_{\text{row}}}$. Fix a malicious prover who sends the value $\vec{b}'$. Then one of the following three cases happens.

- The verifier samples r such that $\sum_{i=1}^h G_{\mathsf{sub}}(r)U_i\vec{x}_i = \vec{0} \mod H^{n_{\text{row}}}$. This case happens with probability at most ϵ_{sub} by Definition 5.3.
- $\vec{b}' \neq \vec{0} \mod H^\lambda$. In this case the verifier always rejects and so the soundness error is 0.
- $\vec{b}' \neq \sum_{i=1}^h G_{\mathsf{sub}}(r)U_i\vec{x}_i$. In this case, the multi-lincheck protocol guarantees a soundness error of ϵ_{LC}.

Thus the soundness error is the maximum of the three.

Prover arithmetic complexity. First, the prover computes $(B_i)_{i \in [h]}$ by multiplying the matrices $U_i = \sum_{j \in [b]} A_{ij}e_j$ with the matrix $R \in \mathbb{F}_p^{\lambda \times n_{\text{row}}}$. So in total this procedure takes $O(\lambda \cdot M_{\mathsf{E}})$ $\mathbb{F}_p$-operations. On top of this step, the prover also computes $\vec{b}$, which is done by multiplying B_is with $\vec{x}_i$s. Thus this step takes $O(h \cdot \lambda n_{\mathrm{col}})$ $\mathbb{F}$-operations. The other prover operations such as computing $G_{\mathsf{sub}}(r)$, and engaging in the multi-lincheck protocol take $\mathsf{o}_{G_{\mathsf{sub}}}$ $\mathbb{F}_p$-operations and $\mathsf{tp}_{\mathrm{LC}}$-$\mathbb{F}$ operations.

Verifier arithmetic complexity. Checking that $\vec{b} = \vec{0} \mod H^\lambda$ takes $O(\lambda \cdot \log_p(|\mathbb{F}|)^2 \cdot (\log_p(|\mathbb{F}|) - \dim(H)))$ $\mathbb{F}_p$-operations. As shown above, computing R and B_is takes $O(\lambda \cdot M_{\mathsf{E}} + \mathsf{o}_{G_{\mathsf{sub}}})$ $\mathbb{F}_p$-operations, and engaging in the multi-lincheck protocol take $\mathsf{tv}_{\mathrm{LC}}$ $\mathbb{F}$-operations.

Other parameters. The other parameters follow directly from the construction. □

6.3 A tensor IOP for R1CS

In this part, we construct a tensor IOP for R1CS from the modular multi-lincheck protocol and the twisted scalar-product protocol. Write $\epsilon_{\mathrm{Mlin_h}}$ for the soundness error of the modular multi-lincheck protocol, $\mathsf{r}_{\mathrm{Mlin_h}}$ for its round complexity, $\mathsf{l}_{\mathrm{Mlin_h}}$ for its proof length, $\mathsf{c}_{\mathrm{Mlin_h}}$ for its communication complexity, $\mathsf{tp}_{\mathrm{Mlin_h}}$ for its prover operations, $\mathsf{tv}_{\mathrm{Mlin_h}}$ for its verifier operations, and $\mathsf{rd}_{\mathrm{Mlin_h}}$ for its randomness complexity. Use $\epsilon_{\mathrm{TSP}}, \mathsf{r}_{\mathrm{TSP}}, \mathsf{l}_{\mathrm{TSP}}, \mathsf{c}_{\mathrm{TSP}}, \mathsf{tp}_{\mathrm{TSP}}, \mathsf{tv}_{\mathrm{TSP}}, \mathsf{rd}_{\mathrm{TSP}}$ to denote the corresponding parameters of the twisted scalar-product protocol. Define o_G to be the arithmetic complexity of computing ϵ-biased generator G over $\mathbb{F}_{p^b}$.

[Notation could be better defined. e.g. in the soundness error, ϵ_{SP} needs to be defined for a particular vector length over a particular field —*Jonathan*] **[Is there a particular reason that we use SP not TSP? I don't think SP protocol is defined. —*Ziyi*]** ★ ★

Theorem 6.11. *For any prime power p, integer $a \in \Omega(\log_p(\log n_{\mathrm{col}}))$, and positive integers k, t, given a systematic $(a,b)_p$-RMFE (ϕ, ψ) with basis E that embeds vectors in $\mathbb{F}_p^a$ to elements in $\mathbb{F}_{p^b}$ (where $b = O(a)$), and an ϵ-biased generator $G\colon S^s \to \mathbb{F}^k$, there is a $(\mathbb{F}_{p^b}, k, t)$-tensor IOP, with non-adaptive queries, for the indexed relation R_{R1CS} that supports instances over $\mathbb{F}_p$ with $n_{\mathrm{row}} = n_{\mathrm{col}} = a \cdot k^t$, $n_{\mathrm{in}} = a \cdot \ell_{\mathrm{in}} \cdot k^{t_{\mathrm{in}}}$, and has the following parameters:*

- *soundness error is $\max(t\epsilon, \epsilon_{\mathrm{Mlin_h}}, \epsilon_{\mathrm{TSP}})$;*
- *round complexity is $O(\max(\mathsf{r}_{\mathrm{Mlin_h}}, \mathsf{r}_{\mathrm{TSP}}))$;*
- *proof length is $O(n_{\mathrm{row}}/a + \mathsf{l}_{\mathrm{Mlin_h}} + \mathsf{l}_{\mathrm{TSP}})$ elements in $\mathbb{F}_{p^b}$;*
- *query complexity is $O(\mathsf{qc}_{\mathrm{Mlin_h}} + \mathsf{qc}_{\mathrm{TSP}})$;*
- *the prover sends $O(\mathsf{c}_{\mathrm{Mlin_h}} + \mathsf{c}_{\mathrm{TSP}})$ non-oracle messages over $\mathbb{F}_{p^b}$;*
- *the prover uses $O(M + n_{\mathrm{row}})$ $\mathbb{F}_p$-operations and $O(t \cdot (\mathsf{o}_G + n_{\mathrm{row}}/a))$ $\mathbb{F}_{p^b}$-operations in addition to the $O(\mathsf{tp}_{\mathrm{Mlin_h}} + \mathsf{tp}_{\mathrm{TSP}})$ operations from the sub-protocols;*
- *the verifier uses $O(M + n_{\mathrm{row}})$ $\mathbb{F}_p$-operations and $O(t \cdot \mathsf{o}_G)$ $\mathbb{F}_{p^b}$-operations in addition to the $O(\mathsf{tv}_{\mathrm{Mlin_h}} + \mathsf{tv}_{\mathrm{TSP}})$ operations from the sub-protocols;*
- *the verifier has randomness complexity $O(ts)$ over S in addition to the $O(\mathsf{rd}_{\mathrm{Mlin_h}} + \mathsf{rd}_{\mathrm{TSP}})$ randomness for the sub-protocols.*

Here M denotes the number of non-zero entries in the instance matrices A, B, C.

The strategy here is to embed a R1CS instance over $\mathbb{F}_p$ into a larger field $\mathbb{F}_{p^b}$ of cardinality $\left|\mathbb{F}_{p^b}\right| = p^{O(a)}$. Then the R1CS relation for matrices over $\mathbb{F}_p$ is translated into a relation over $\mathbb{F}_{p^b}$ as stated in the following lemma.

Lemma 6.12 ([CascudoG21])**.** *Let $(\mathbb{i}, \mathbb{x}) = \big((\mathbb{F}_p, n_{\mathrm{row}}, n_{\mathrm{col}}, A, B, C), (n_{\mathrm{in}}, x)\big)$ be a* R1CS *index-instance pair, and let (ϕ, ψ) be a systematic $(a,b)_p$-RMFE. Then there exists $\mathbb{w} \in \mathbb{F}_p^{n_{\mathrm{col}} - n_{\mathrm{in}}}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in R_{\mathrm{R1CS}}$ if and only if there exists $\widetilde{z} \in \mathbb{F}_{p^b}^{n_{\mathrm{col}}/a}$ and $\widetilde{z_A}, \widetilde{z_B}, \widetilde{z_C}, \widetilde{b} \in \mathbb{F}_{p^b}^{n_{\mathrm{row}}/a}$ satisfying* **[*clash* b, b *—Jonathan*]** ★

$$\widetilde{z_A} \circ \widetilde{z_B} = \widetilde{b} \tag{3}$$

$$\widetilde{z} = \vec{0} \quad \bmod\ (\operatorname{Im}\phi)^{n_{\text{col}}/a} \tag{4}$$
$$\widetilde{z_U} = \vec{0} \quad \bmod\ (\operatorname{Im}\phi)^{n_{\text{row}}/a} \qquad \forall U = A, B, C \tag{5}$$
$$\widetilde{U}\widetilde{z} - \widetilde{I_{n_{\text{row}}}}\widetilde{z_U} = \vec{0} \quad \bmod\ \left(\ker \sum_{j=1}^{a} \star\psi\right)^{n_{\text{row}}} \qquad \forall U = A, B, C \tag{6}$$
$$\widetilde{b} - u\widetilde{z_C} = \vec{0} \quad \bmod\ (\ker\psi)^{n_{\text{row}}/a} \tag{7}$$

where $z = (x, \mathbb{w})$, $\widetilde{z_U} = \Phi(Uz)$, $\widetilde{z} = \Phi(z)$, $\widetilde{U} = \Phi(U)$ and $u = \phi(\mathbf{1}^a)$. We use the notation $\Phi(U)$ to denote the matrix obtained by applying ϕ row-wise to the matrix U, and the notation $\sum_{j=1}^{a} \star\psi$ to denote the operation of applying ψ to an element in $\mathbb{F}_{p^b}$ and then sum up the entries of the resulting vector in $\mathbb{F}_p^a$.

Now we construct an interactive oracle proof $\mathsf{IOP} = (\mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation R_{R1CS} by checking whether the above relation holds over the larger field $\mathbb{F}_{p^b}$. ↩ DOUBLE EMBEDDINGS

Construction 6.13 (tensor IOP for R1CS)**.** The prover $\mathbf{P}$ takes as input $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}_p, n_{\text{row}}, n_{\text{col}}, A, B, C), (n_{\text{in}}, x), w\big)$, while the verifier $\mathbf{V}$ takes as input $(\mathbb{i}, \mathbb{x})$.

- The prover $\mathbf{P}$ constructs the full assignment $z := (x, w) \in \mathbb{F}_p^{n_{\text{col}}}$ and computes the vectors

  $$\widetilde{z} := \Phi(z) \in \mathbb{F}_{p^b}^{n_{\text{col}}/a}, \qquad \widetilde{z_A} := \Phi(Az), \qquad \widetilde{z_B} := \Phi(Bz), \qquad \widetilde{z_C} := \Phi(Cz), \qquad \widetilde{b} := \widetilde{z_A} \circ \widetilde{z_B} \in \mathbb{F}_{p^b}^{n_{\text{row}}/a}$$

  The prover sends the oracle message $\Pi_1 := (\widetilde{z}, \widetilde{z_A}, \widetilde{z_B}, \widetilde{z_C}, \widetilde{b})$.

- The verifier $\mathbf{V}$ sends uniformly random seeds $\rho_1, \ldots, \rho_t \in S^s$.

- The prover $\mathbf{P}$ computes the query vector $r := G(\rho_1) \otimes \cdots \otimes G(\rho_t)$, and the field element $\nu_b := \langle r, \widetilde{b}\rangle$. The prover $\mathbf{P}$ sends the non-oracle message $\nu_b \in \mathbb{F}_{p^b}$.

- The verifier $\mathbf{V}$ performs consistency checks. First, $\mathbf{V}$ queries $\widetilde{b}$ in Π_1 at $r = G(\rho_1) \otimes \cdots \otimes G(\rho_t)$ in order to obtain the answer $\langle r, \widetilde{b}\rangle$. Then, $\mathbf{V}$ checks that $\nu_b = \langle r, \widetilde{b}\rangle$, which shows that ν_b is the correct answer to the query on $\widetilde{b}$.

  Moreover, $\mathbf{V}$ checks that the (claimed) embedded satisfying assignment $\widetilde{z}$ is consistent with the partial assignment x as follows: sample uniformly random seeds $\sigma_1, \ldots, \sigma_{t_{\text{in}}+1} \in S^s$; compute vectors $s_i = G(\sigma_i)$ for $i \in [t_{\text{in}}]$, and compute $s_{t_{\text{in}}+1}$ by computing $G(\sigma_{t_{\text{in}}+1})$ and changing all but the first ℓ_{in} entries to zero. Set $s'_{t_{\text{in}}+1}$ to be the first ℓ_{in} entries. Then set each of the vectors $s_{t_{\text{in}}+2}, \ldots, s_t \in \mathbb{F}_{p^b}^k$ to equal $(1, 0, \ldots, 0) \in \mathbb{F}_{p^b}^k$; query $\widetilde{z}$ in Π_1 at the tensor $s := s_1 \otimes \cdots \otimes s_t$ in order to obtain the answer $\langle s, \widetilde{z}\rangle$; and check that $\langle s, \widetilde{z}\rangle = \langle s_1 \otimes \cdots \otimes s_{t_{\text{in}}} \otimes s'_{t_{\text{in}}+1}, \widetilde{x}\rangle$.

- The prover $\mathbf{P}$ and verifier $\mathbf{V}$ engage in several sub-protocols to check equations (1)-(5) of Lemma 6.12.
  - A twisted scalar-product protocol with instance $\mathbb{x} = (\mathbb{F}_{p^b}, n_{\text{row}}/a, r, \nu_b)$ and witness $\mathbb{w} = (\widetilde{z_A}, \widetilde{z_B})$ to show that $\langle \widetilde{z_A} \circ r, \widetilde{z_B}\rangle = \nu_b$.
  - For every $U \in \{A, B, C\}$, a modular multi-lincheck protocol with $\mathbb{i} = (\operatorname{Im}\phi, I_{n_{\text{row}}/a})$, $\mathbb{x} = (\mathbb{F}_{p^b}, \mathsf{E}, n_{\text{row}}/a, n_{\text{row}}/a, 1)$, and $\mathbb{w} = \widetilde{z_U}$ to show that $\widetilde{z_U} = \vec{0} \ \bmod\ (\operatorname{Im}\phi)^{n_{\text{row}}/a}$.
  - For every $U \in \{A, B, C\}$, a modular multi-lincheck protocol with $\mathbb{i} = (\ker \sum_{j=1}^{n_{\text{col}}/a} \star\phi, (\widetilde{U}, -\widetilde{I_{n_{\text{row}}}}))$, $\mathbb{x} = (\mathbb{F}_{p^b}, \mathsf{E}, n_{\text{row}}, n_{\text{col}}/a, 2)$, and $\mathbb{w} = (\widetilde{z}, \widetilde{z_U})$ to show that $\widetilde{U}\widetilde{z} - \widetilde{I_{n_{\text{row}}}}\widetilde{z_U} = \vec{0} \ \bmod\ (\ker \sum_{j=1}^{n_{\text{col}}/a} \star\phi)^{n_{\text{row}}}$.

– A modular multi-lincheck protocol with $\mathbb{i} = (\ker \psi, (I_{n_{\mathrm{row}}/a}, I_{n_{\mathrm{row}}/a}))$, $\mathbb{x} = (\mathbb{F}_{p^b}, \mathsf{E}, n_{\mathrm{row}}/a, n_{\mathrm{row}}/a, 2)$, and $\mathbb{w} = (\widetilde{b}, -u\widetilde{z_C})$ to show that $\widetilde{b} - u\widetilde{z_C} = \vec{0} \mod (\ker \psi)^{n_{\mathrm{row}}/a}$.

Proof. **Completeness.** Let $z := (x, \mathbb{w})$ be a solution to the R1CS instance. The honest prover sends the oracles $\widetilde{z} := \Phi(z)$, $\widetilde{z_U} := \Phi(Uz)$ and $\widetilde{b} := \widetilde{z_A} \circ \widetilde{z_B}$, and the non-oracle message $\nu_b := \langle r, \widetilde{b}\rangle$. It implies that we have $\nu_b = \langle \widetilde{z_A} \circ r, \widetilde{z_B}\rangle$, and that all the modular equations in Lemma 6.12 are satisified. This means that the twisted scalar-product protocol and the modular multi-lincheck protocols succeed. Finally by construction, $\widetilde{z}$ is consistent with the partial assignment x. So it always holds that $\langle s, \widetilde{z}\rangle = \langle s_1 \otimes \cdots \otimes s_{t_{\mathrm{in}}} \otimes s'_{t_{\mathrm{in}}+1}, \widetilde{x}\rangle$.

Soundness error. Let $\mathbb{x} = (\mathbb{F}_p, n_{\mathrm{row}}, n_{\mathrm{col}}, A, B, C, x)$. Fix a malicious prover who sends oracle messages $\Pi_1 := (\widetilde{z}, \widetilde{z_A}, \widetilde{z_B}, \widetilde{z_C}, \widetilde{b})$. Then one of the following cases must hold.

- $\widetilde{z}$ is not consistent with the embedded partial assignment $\widetilde{x}$. In this case the verifier samples s such that $\langle s, \widetilde{z}\rangle = \langle s_1 \otimes \cdots \otimes s_{t_{\mathrm{in}}} \otimes s'_{t_{\mathrm{in}}+1}, \widetilde{x}\rangle$ with probability at most $(t_{\mathrm{in}} + 1)\epsilon$ by the property of the ϵ-biased generator.
- $\widetilde{b} \neq \widetilde{z_A} \circ \widetilde{z_B}$. In this case the verifier samples r such that $\langle r, \widetilde{b}\rangle = \langle \widetilde{z_A} \circ r, \widetilde{z_B}\rangle$ with probability at most $t\epsilon$ by the property of the ϵ-biased generator. If this equality does not hold, then the twisted scalar-product protocol gives the soundness error guarantee of ϵ_{SP}. Therefore the soundness error in this case is $\max(t\epsilon, \epsilon_{\mathrm{SP}})$.
- At least one equation among Equations (2)-(5) in Lemma 6.12 does not hold. Then the modular lincheck protocols give the soundness error guarantee of $\epsilon_{\mathrm{Mlin_h}}$.

Therefore altogether the protocol has soundness error $\max(t\epsilon, \epsilon_{\mathrm{SP}}, \epsilon_{\mathrm{Mlin_h}})$.

Prover arithmetic complexity. In step 1, the prover computes Uz for $U = A, B, C$, encodes the resulting vectors, and finally computes the entry-wise product $\widetilde{b}$. The first part takes $O(M)$ operations in $\mathbb{F}_p$, the second part takes $O(3n_{\mathrm{row}} + n_{\mathrm{col}})$ operations in $\mathbb{F}_p$, and the last part takes $O(n_{\mathrm{row}}/a)$ operations in $\mathbb{F}_{p^b}$. In step 3, the prover computes the query vector r and the field element ν_b. In total this step takes $O(t(\mathsf{o}_G + n_{\mathrm{row}}/a))$ $\mathbb{F}_{p^b}$-operations. In step 5, the prover encodes the matrices A, B, C and $I_{n_{\mathrm{row}}}$ and represents them in the basis E before running the subprotocols. Since the encoding ϕ is systematic, the encoding time is linear in the number of non-zero elements in the matrices. So this part takes $O(M + n_{\mathrm{row}})$ operations in $\mathbb{F}_p$ in addition to the operations in the subprotocols.

Verifier arithmetic complexity. In step 4, the verifier computes query vectors $(G(\rho_i))_{i\in[t]}$ and $(s_i)_{i\in[t]}$, which takes $O(t \cdot \mathsf{o}_G)$ operations in $\mathbb{F}_{p^b}$. In step 5, the verifier encodes the matrices A, B, C and $I_{n_{\mathrm{row}}}$ before running the subprotocols, which takes $O(M + n_{\mathrm{row}})$ operations in $\mathbb{F}_p$ in addition to the operations in the subprotocols.

Other parameters. The other parameters follow directly from the construction. □

7 Algebraic automata

In this section, we focus on constructing a tensor IOP for R1CS automata, a relation that is similar to but more structured than the R1CS relation. We slightly modify the notion of R1CS automata introduced in [**BenSassonCGGRS19**] and explain how we adapt our tensor IOP for R1CS to construct an IOP for R1CS automata achieving sublinear verifier while preserving the linear-time prover. The construction relies on a cyclic-shift test that enables a multi-lincheck protocol with sublinear verification, which is the bottleneck for the verification time in the IOP for R1CS.

Definition 7.1. *The* R1CS *automata relation R_{R1CSA} is the set of triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (x, (\mathbb{F}_p, A, B, C), (w, T), z) \enspace .$$

Here, $\mathbb{F}_p$ is a finite field, $T \in \mathbb{N}$ is the computation time of the automata, and w is the computation width, $A, B, C \in \mathbb{F}_p^{w \times 2w}$ define the time constraints, $x \in \mathbb{F}_p^w$ defines boundary conditions, and the execution trace $z \colon [T+1] \to \mathbb{F}_p^w$ is a function that specifies the content of the w registers at each time step. The trace satisfies $z(1) = x$, and for all $t \in [T]$, $A(z(t), z(t+1)) \circ B(z(t), z(t+1)) = C(z(t), z(t+1))$. (Here "$\circ$" is the entry-wise product and "$(\cdot, \cdot)$" is the vector concatenation.)

Theorem 7.2. *For every prime power p, and every finite field $\mathbb{F}_{p^b}$ such that $p^b \in \Omega(\log n_{\mathrm{col}})$, given a systematic $(a, b)_p$-RMFE (ϕ, ψ) such that $a \in \Theta(b)$, there is a $(\mathbb{F}_{p^b}, k, t)$-tensor IOP, with non-adaptive queries, for the indexed automata relation R_{R1CSA} that supports instances over $\mathbb{F}_p$ with computation width w, computation time T, $(T+1)w = a \cdot k^t$ and has the following parameters:*
- *soundness error is $O(1)$;*
- *round complexity is $O(\log(wT/a))$;*
- *proof length is $O(wT/a)$ elements in $\mathbb{F}_{p^b}$ and $O(wT/a)$ elements in $\mathbb{F}_p$;*
- *query complexity is $O(t)$;*
- *the prover sends $O(\log(wT/a))$ non-oracle messages over $\mathbb{F}_{p^b}$;*
- *the prover uses $O(w^2 T)$ $\mathbb{F}_p$-operations and $O(twT/a))$ $\mathbb{F}_{p^b}$-operations;*
- *the verifier uses $O(b^3)$ $\mathbb{F}_p$-operations, $O(t(wT)^{1/t})$ $\mathbb{F}_{p^\lambda}$-operations, and $O(tk)$ $\mathbb{F}_{p^b}$-operations;*
- *the verifier has randomness complexity $O(\log(wT/a))$ over $\mathbb{F}_{p^b}$ and $O(\log(wT/a))$ over $\mathbb{F}_p$.*

7.1 R1CS automata

As described in Section 2.6, in order to adapt the techniques we use when designing IOPs for R1CS, we express the automata relation in terms of the R1CS relation. To do that, we rewrite the register-wise Hadamard product constraints as a single matrix-vector-multiplication Hadamard product constraint by exploiting the special structure of *staircase matrices* as described below.

Definition 7.3. *The* **rectangular identity matrix** *$I_{T \times (T+1)} \in \mathbb{F}^{T \times (T+1)}$ is defined as*

$$\vec{I}_T = \begin{pmatrix} 1 & 0 & & & \\ & 1 & 0 & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{pmatrix} .$$

Definition 7.4. *The* **shifted identity matrix** $\overrightarrow{I}_{T\times(T+1)} \in \mathbb{F}^{T\times(T+1)}$ *is defined as*

$$\overrightarrow{I}_T = \begin{pmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \end{pmatrix}.$$

Definition 7.5. *The* **staircase matrix** *of two matrices $M, M' \in \mathbb{F}^{w\times w}$ is the matrix in $\mathbb{F}^{wT\times w(T+1)}$ defined as*

$$S(M, M') = I_{T\times(T+1)} \otimes M + \overrightarrow{I}_{T\times(T+1)} \otimes M' = \begin{pmatrix} M & M' & & & & \\ & M & M' & & & \\ & & M & M' & & \\ & & & \ddots & \ddots & \\ & & & & M & M' \end{pmatrix}$$

where $I_{T\times(T+1)}$ is the rectangular identity matrix and $\overrightarrow{I}_{T\times(T+1)}$ is the shifted identity matrix.

Lemma 7.6 (Reduction from R_{R1CSA} to staircase matrices)**.** *For any $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = ((\mathbb{F}_p, A, B, C), (w, T), z)$, consider $S_A = S(A_1, A_2), S_B = S(B_1, B_2), S_C = S(C_1, C_2) \in \mathbb{F}_p^{wT\times w(T+1)}$ where M_1 is the first w columns of M and M_2 is the remaining w columns for all $M \in \{A, B, C\}$. Viewing z as a vector in $\mathbb{F}_p^{w(T+1)}$, $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in R_{\mathrm{R1CSA}}$ iff $S_A z \circ S_B z = S_C z$.*

Proof. Note that we have

$$\begin{aligned} S_A z = S(A_1, A_2) \cdot z = (A(z(1), z(2)), A(z(2), z(3)), \cdots, A(z(T), z(T+1)), \\ S_B z = S(B_1, B_2) \cdot z = (B(z(1), z(2)), B(z(2), z(3)), \cdots, B(z(T), z(T+1)), \\ S_C z = S(C_1, C_2) \cdot z = (C(z(1), z(2)), C(z(2), z(3)), \cdots, C(z(T), z(T+1)). \end{aligned}$$

The lemma follows. □

As in Section 6, we need to embed the instance matrices $S_A, S_B, S_C \in \mathbb{F}_p^{wT\times w(T+1)}$ as defined in Lemma 7.6 with a systematic $(a,b)_p$-RMFE (ϕ, ψ) to $\widetilde{S_A}, \widetilde{S_B}, \widetilde{S_C} \in \mathbb{F}_{p^b}^{wT/a\times w(T+1)}$ such that w divides a.

Definition 7.7. *For any staircase matrix $S(M, M') \in \mathbb{F}_p^{wT\times w(T+1)}$ and any systematic $(a,b)_p$-RMFE such that w divides a, we define the* ***embedded staircase matrix*** *to be*

$$\widetilde{S(M, M')} = \begin{pmatrix} m_0 & m_1 & & & & \\ & m_0 & m_1 & & & \\ & & m_0 & m_1 & & \\ & & & \ddots & \ddots & \\ & & & & m_0 & m_1 \end{pmatrix} \in \mathbb{F}_{p^b}^{wT\times(wT/a+1)}$$

where $m_0, m_1 \in \mathbb{F}_{p^b}^{a}$ are the ϕ-embeddings of the following two square matrices

$$M_0 = \begin{pmatrix} M & M' & & & \\ & M & M' & & \\ & & M & M' & \\ & & & \ddots & \ddots \\ & & & & M \end{pmatrix} \in \mathbb{F}_p^{a\times a},\ M_1 = \begin{pmatrix} 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \\ M' & 0 & & 0 \end{pmatrix} \in \mathbb{F}_p^{a\times a}$$

Thus $\widetilde{S(M, M')} = I_{Tw/a\times(Tw/a+1)} \otimes m_0 + \overrightarrow{I}_{Tw/a} \otimes m_1$.

So for any instance matrices $U \in \{A, B, C\}$, the embedded staircase matrix $\widetilde{S_U}$ has a simple decomposition into to two tensors.

7.2 Achieving sublinear verification for R1CS automata

Next we extend the tensor IOP for R1CS in Section 2.4 to achieve sublinear verification for the R1CS automata relation R_{R1CSA}.

In the current construction Construction 6.13, the verifier needs to perform matrix-matrix calculations in the modular multi-lincheck protocols. To reduce the arithmetic complexity of the verifier, we make the modular multi-lincheck protocol in Construction 6.10 to have sublinear verifier by making the prover prove the correctness of the calculations to the verifier via a tensor-product protocol.

In the rest of the section we describe the construction of a modular lincheck protocol with sublinear verification, in which we use several sub-protocols, including the cyclic shift protocol and the biased generator consistency that are mentioned in Section 2.6 and will be discussed thoroughly in later sections.

We write $\epsilon_{\mathrm{CS}}, \epsilon_{\mathrm{TSP}}, \epsilon_{\mathrm{BGC}}$ for the soundness error of the cyclic shift protocol, the twisted scalar-product protocol, and the biased generator consistency test, $\mathsf{r}_{\mathrm{CS}}, \mathsf{r}_{\mathrm{TSP}}, \mathsf{r}_{\mathrm{BGC}}$ for their round complexity, $\mathsf{l}_{\mathrm{CS}}, \mathsf{l}_{\mathrm{TSP}}, \mathsf{l}_{\mathrm{BGC}}$ for their proof length, $\mathsf{qc}_{\mathrm{CS}}, \mathsf{qc}_{\mathrm{TSP}}, \mathsf{qc}_{\mathrm{BGC}}$ for their query complexity, $\mathsf{c}_{\mathrm{CS}}, \mathsf{c}_{\mathrm{TSP}}$ for communication complexity, $\mathsf{tp}_{\mathrm{CS}}, \mathsf{tp}_{\mathrm{TSP}}, \mathsf{tp}_{\mathrm{BGC}}$ for prover arithmetic complexity, $\mathsf{tv}_{\mathrm{CS}}, \mathsf{tv}_{\mathrm{TSP}}, \mathsf{tv}_{\mathrm{BGC}}$ for verifier arithmetic complexity, and $\mathsf{rd}_{\mathrm{CS}}, \mathsf{rd}_{\mathrm{TSP}}, \mathsf{rd}_{\mathrm{BGC}}$ for randomness complexity. We also use $\mathsf{o}_{G_{\mathsf{sub}}}$ to denote the arithmetic complexity of computing G_{sub}.

Lemma 7.8 (Special case of Theorem 6.9)**.** *For every prime power p, and every finite field $\mathbb{F}_{p^b}$, every $\mathbb{F}_p$-linear subspace $H \subseteq \mathbb{F}_{p^b}$, and positive integers k, t, given a ϵ-biased generator $G\colon \mathbb{F}_{p^\lambda}^s \to \mathbb{F}_{p^\lambda}^{(wT/\lambda t)^{1/t}}$, there is a $(\mathbb{F}_{p^b}, k, t)$-tensor IOP, with non-adaptive queries, for the indexed relation $R_{\mathrm{Mlin_h}}$ that supports staircase instances over $\mathbb{F}_{p^b}$ with block size w, $n_{\mathrm{row}} = wT$, and $n_{\mathrm{col}} = \ell \cdot k^t$, and has the following parameters:*

- *soundness error is $\max(t\epsilon, \lambda \log \frac{n_{\mathrm{row}}}{a}/p^b, \epsilon_{\mathrm{CS}}, \epsilon_{\mathrm{BGC}})$;*
- *round complexity is $O(\lambda \mathsf{r}_{\mathrm{TSP}} + \mathsf{r}_{\mathrm{CS}} + \mathsf{r}_{\mathrm{BGC}})$;*
- *proof length is $O(h \cdot n_{\mathrm{col}} + h \cdot \mathsf{l}_{\mathrm{CS}} + h \cdot \lambda^2 \cdot \mathsf{l}_{\mathrm{TSP}})$ elements in $\mathbb{F}_{p^b}$ and $O(\mathsf{l}_{\mathrm{BGC}})$ elements in $\mathbb{F}_p$;*
- *query complexity is $O(h \cdot \mathsf{qc}_{\mathrm{CS}} + h \cdot \lambda^2 \cdot \mathsf{qc}_{\mathrm{TSP}} + \mathsf{qc}_{\mathrm{BGC}})$;*
- *the prover sends $O(h \cdot \lambda + h \cdot \mathsf{c}_{\mathrm{CS}} + h \cdot \lambda^2 \cdot \mathsf{c}_{\mathrm{TSP}})$ non-oracle messages;*
- *the prover uses $O(\lambda \cdot M_{\mathsf{E}} + \mathsf{o}_{G_{\mathsf{sub}}})$ $\mathbb{F}_p$-operations, $O(h \cdot \lambda n_{\mathrm{col}} + h \cdot \mathsf{tp}_{\mathrm{CS}} + h \cdot \lambda^2 \cdot \mathsf{tp}_{\mathrm{TSP}})$ $\mathbb{F}_{p^b}$-operations, and $O(\mathsf{tp}_{\mathrm{BGC}})$ $\mathbb{F}_{p^\lambda}$-operations;*
- *the verifier uses $O(\lambda \cdot b^3)$ $\mathbb{F}_p$-operations, $O(h \cdot \mathsf{tv}_{\mathrm{CS}} + h \cdot \lambda^2 \cdot \mathsf{tv}_{\mathrm{TSP}})$ $\mathbb{F}_{p^b}$-operations and $O(\mathsf{tv}_{\mathrm{BGC}})$ $\mathbb{F}_{p^\lambda}$-operations;*
- *the verifier has randomness complexity $O(ts)$ over $\mathbb{F}_{p^\lambda}$, $O(h \cdot \mathsf{rd}_{\mathrm{CS}} + h \cdot \lambda^2 \cdot \mathsf{rd}_{\mathrm{TSP}})$ elements in $\mathbb{F}_{p^b}$, and $O(\mathsf{rd}_{\mathrm{BGC}})$ elements in $\mathbb{F}_p$.*

Here M_{E} denotes the number of non-zero entries in U_is' coefficient matrices (notations as defined in Definition 6.8).

The construction mostly follows the steps in Construction 6.10. The only changes are the following. For a randomly generated vector $\vec{r}$ the verifier needs to check that $\vec{r}^{\mathsf{T}} S_U \vec{x} = \tau$ where $S_U \in \{\widetilde{S_A}, \widetilde{S_B}, \widetilde{S_C}, \widetilde{I_{n_{\mathrm{row}}}}, \widetilde{I_{n_{\mathrm{row}}/a}}\}$ are embedded staircase matrice. We note that for any embedded staircase matrix $\widetilde{S(M, M')}$ the left hand of the equation can be written as

$$\vec{r}^\intercal \widetilde{S(M,M')}\vec{x} = \langle \vec{r} \circ (\vec{1}_{Tw/a} \otimes m_0) \circ (\vec{x}_0 \otimes \vec{1}_a) + \vec{r} \circ (\vec{1}_{Tw/a} \otimes m_1) \circ (\vec{x}_1 \otimes \vec{1}_a), \vec{1}_{Tw} \rangle \tag{8}$$

where $\vec{x}_0 = \vec{x}[1 : Tw/a]$ and $\vec{x}_1 = \vec{x}[2 : Tw/a + 1]$.

Now we present the construction.

Construction 7.9 (tensor IOP for $R_{\mathrm{Mlin_h}}$ with sublinear verification)**.** We construct an interactive oracle proof $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation $R_{\mathrm{Mlin_h}}$ that achieves sublinear verification.

Given the index $\mathbb{i} = (H, (U_i)_{i\in[h]})$, the indexer $\mathbf{I}$ computes and outputs as oracle message $\Pi_0 = (U_{i,0}, U_{i,1})_{i\in[h]}$, where $U_i = \widetilde{S(M_i, M_i')}$ are the embedded staircase matrices in Definition 7.7, $U_{i,0} = \vec{1}_{Tw/a} \otimes m_0$, and $U_{i,1} = \vec{1}_{Tw/a} \otimes m_1$.

The prover $\mathbf{P}$ takes as input the index $\mathbb{i}$, instance $\mathbb{x} = (\mathbb{F}, \mathsf{E}, n_{\text{row}}, n_{\text{col}}, h)$, and witness $\mathbb{w} = (\vec{x}_i)_{i\in[h]}$, while the verifier takes as input Π_0 and $\mathbb{x}$.

- $\mathbf{P}$ computes and sends the oracle message $(\vec{x}_{i,0}, \vec{x}_{i,1})_{i\in[h]}$ where $\vec{x}_{i,0}, \vec{x}_{i,1}$ are as defined in Equation (8).
- The verifier $\mathbf{V}$ sends uniformly random seeds $r \in \mathbb{F}_{p^\lambda}^{st}$.
- The prover uses the biased generator G_{sub} defined in Construction 7.16 to generate the matrix $R = G_{\mathsf{sub}}(r) \in \mathbb{F}_p^{\lambda\times n_{\text{row}}}$ and then computes the vector $\vec{b} = \sum_{i=1}^{h} RU_i\vec{x}_i \in \mathbb{F}_{p^b}^{\lambda}$ and sends the non-oracle message $\vec{b}$.
- The verifier first checks that $\vec{b} = \vec{0} \mod H^\lambda$, and rejects if the check fails.
- The prover computes $\vec{b}_i = RU_i\vec{x}_i$ for all $i \in [h]$, and sends $(\vec{b}_i)_{i\in[h]}$ as non-oracle message and $(R_j)_{j\in[\lambda]}$ as oracles to the verifier.
- The verifier checks that $\vec{b} = \sum_{i=1}^{h} \vec{b}_i$.
- For every $i \in [h]$, the prover and the verifier engage in a cyclic shift protocol to check that $\vec{x}_{i,1}$ is the 1-entry cyclic shift of $\vec{x}_{i,0}$ over $\mathbb{F}_{p^b}$. This relation and its protocol are explained in more detail in Section 7.3.
- For every $i \in [h]$ and $j \in [\lambda]$, the prover and the verifier engage in 2λ TSP protocols to check that $\langle R_j \circ (\vec{x}_{i,0} \otimes \vec{1}_a), U_{i,0}\rangle + \langle R_j \circ (\vec{x}_{i,1} \otimes \vec{1}_a), U_{i,1}\rangle = (\vec{b}_i)_j$ (see Remark 7.10).
- The prover and the verifier enagage in the protocol from Construction 7.19 to check that $R = G_{\mathsf{sub}}(r)$.

Remark 7.10. By construction of G_{sub}, the j-th row of R has the following tensor structure (see Section 7.4 for more on the structure of R)

$$R_j = \sum_{\ell_{t-1},\ell_t\in[\lambda]} u^{(j,\ell_{t-1})} \otimes e^{(\ell_{t-1},\ell_t)} \quad ,$$

where $u^{(j,\ell_{t-1})} \in \mathbb{F}_p^{Tw/a}$ and $e^{(\ell_{t-1},\ell_t)} \in \mathbb{F}_p^a$. Plugging this decomposition of R_j into the expression, we obtain that

$$\begin{aligned}&\langle R_j \circ (\vec{x}_{i,0} \otimes \vec{1}_a), U_{i,0}\rangle + \langle R_j \circ (\vec{x}_{i,1} \otimes \vec{1}_a), U_{i,1}\rangle \\ &= \sum_{\ell_{t-1}\in[\lambda]} \langle u^{(j,\ell_{t-1})} \circ \vec{x}_{i,0}, \vec{1}_{Tw/a}\rangle \cdot \sum_{\ell_t\in[\lambda]} \langle e^{(\ell_{t-1},\ell_t)} \circ \vec{1}_a, m_0\rangle + \sum_{\ell_{t-1}\in[\lambda]} \langle u^{(j,\ell_{t-1})} \circ \vec{x}_{i,1}, \vec{1}_{Tw/a}\rangle \cdot \sum_{\ell_t\in[\lambda]} \langle e^{(\ell_{t-1},\ell_t)} \circ \vec{1}_a, m_1\rangle\end{aligned}$$

The verifier can compute the terms $\sum_{\ell_t \in [\lambda]} \langle e^{(\ell_{t-1},\ell_t)} \circ \vec{1}_a, m_c \rangle$, $c \in \{0,1\}$ $\ell_{t-1} \in [\lambda]$, with $O(\lambda^2 \cdot a)$ operations in $\mathbb{F}_{p^b}$. To check the values for the terms $\langle u^{(j,\ell_{t-1})} \circ \vec{x}_{i,c}, \vec{1}_{Tw/a} \rangle = \langle u^{(j,\ell_{t-1})} \circ \vec{1}_{Tw/a}, \vec{x}_{i,c} \rangle$, $c \in \{0,1\}$, the verifer engages in 2λ TSP protocols checking. By Lemma 6.4, this step takes the prover $O(Tw/a)$ operations in $\mathbb{F}_{p^b}$ and the verifier $O\left(t \sqrt[t-1]{Tw/a}\right)$ operations in $\mathbb{F}_{p^b}$.

Proof. **Completeness.** Completeness is straightforward from the construction.
Soundness error. Suppose $\sum_{i=1}^h U_i \vec{x}_i \neq \vec{0} \mod H^{n_{\text{row}}}$. Fix a malicious prover who sends the messages $\vec{b}'$, $(\vec{x}'_{i,0}, \vec{x}'_{i,1})_{i \in [h]}$,$(\vec{b}'_i)_{i \in [h]}$, and $(R_j)_{j \in [\lambda]}$. Then one of the following six cases happens.

- The verifier samples r such that $\sum_{i=1}^h G_{\mathsf{sub}}(r) U_i \vec{x}_i = \vec{0} \mod H^{n_{\text{row}}}$. This case happens with probability at most ϵ_{sub} by Definition 5.3.
- $\vec{b}' \neq \vec{0} \mod H^\lambda$. In this case the verifier always rejects and so the soundness error is 0.
- $\vec{b}' \neq \sum_{i=1}^h \vec{b}'_i$. In this case the verifier always rejects and so the soundness error is 0.
- For some $i \in [h]$, $\vec{x}_{i,1}$ is not the 1-entry cyclic shift of $\vec{x}_{i,0}$. In this the cyclic shift protocol guarantees a soundness error of at most ϵ_{CS}.
- $\langle R_j \circ (\vec{x}_{i,0} \otimes \vec{1}_a), U_{i,0} \rangle + \langle R_j \circ (\vec{x}_{i,1} \otimes \vec{1}_a), U_{i,1} \rangle \neq \vec{b}'_{i,j}$ for certain i and j. In this case, the TSP protocol guarantees a soundness error of $\epsilon_{\text{TSP}} = \frac{2\lambda \cdot \log \frac{n_{\text{row}}}{a}}{\left|\mathbb{F}_{p^b}\right|}$.
- $R \neq G_{\mathsf{sub}}(r)$. In this case the soundness error is at most ϵ_{BGC}.

Thus the soundness error is the maximum of the six.
Prover arithmetic complexity. First, the prover computes $(\vec{b}_i)_{i \in [h]}$ by multiplying the matrices $U_i = \sum_{j \in [b]} A_{ij} e_j$ with the matrix $R \in \mathbb{F}_p^{\lambda \times n_{\text{row}}}$ and the vector $\vec{x}_i$. So in total this procedure takes $O(\lambda \cdot M_{\mathsf{E}})$ $\mathbb{F}_p$-operations and $O(h \cdot \lambda n_{\text{col}})$ $\mathbb{F}$-operations. Moreover, the prover computes $G_{\mathsf{sub}}(r)$, engages in the cyclic shift protocols, the twisted scalar product protocols, and the test for subspace biased generator, which take $O(\mathsf{o}_{G_{\mathsf{sub}}})$ $\mathbb{F}_p$-operations, $O(h \cdot \mathsf{tp}_{\text{CS}} + h \cdot \lambda^2 \cdot \mathsf{tp}_{\text{TSP}})$ $\mathbb{F}_{p^b}$-operations, and $O(\mathsf{tp}_{\text{BGC}})$ $\mathbb{F}_{p^\lambda}$-operations.
Verifier arithmetic complexity. Checking that $\vec{b} = \vec{0} \mod H^\lambda$ takes $O(\lambda \cdot \log_p(\left|\mathbb{F}_{p^b}\right|)^2 \cdot (\log_p(\left|\mathbb{F}_{p^b}\right|) - \dim(H)))$ $\mathbb{F}_p$-operations. Engaging in the TSP protocols and cyclic shift protocols take $h \cdot \lambda^2 \cdot \mathsf{tv}_{\text{TSP}} + h \cdot \mathsf{tv}_{\text{CS}}$ $\mathbb{F}_{p^b}$-operations. The check for the output of the subspace biased generator takes $O(\mathsf{tv}_{\text{BGC}})$ $\mathbb{F}_{p^\lambda}$-operations.
Other parameters. The other parameters follow directly from the construction. ☐

So now, we can construct a tensor IOP for the R1CS automata relation R_{R1CSA} from the modular multi-lincheck protocol given in Lemma 7.8 as in Section 6.3. Write ϵ_{MLA} for the soundness error of the modular multi-lincheck protocol, r_{MLA} for its round complexity, l_{MLA} for its proof length, c_{MLA} for its communication complexity, tp_{MLA} for its prover operations, tv_{MLA} for its verifier operations, and rd_{MLA} for its randomness complexity. Use $\epsilon_{\text{TSP}}, \mathsf{r}_{\text{TSP}}, \mathsf{l}_{\text{TSP}}, \mathsf{c}_{\text{TSP}}, \mathsf{tp}_{\text{TSP}}, \mathsf{tv}_{\text{TSP}}, \mathsf{rd}_{\text{TSP}}$ to denote the corresponding parameters of the twisted scalar-product protocol. Define o_G to be the arithmetic complexity of computing ϵ-biased generator G over $\mathbb{F}_{p^b}$.

[Make description of parameters consistent with e.g. Theorem 6.2 —*Jonathan*] **[Notation could be better defined. e.g. in the soundness error, ϵ_{SP} needs to be defined for a particular vector length over a particular field —*Jonathan*]** **[in section 6, we didn't define these parameters over specific length/field, should we change them all? —*Ziyi*]** ★ ★ ★

Theorem 7.11. *For every prime power p, integer $a \in \Omega(\log_p n_{\text{col}})$, and positive integers k, t, given a systematic $(a, b)_p$-RMFE (ϕ, ψ) with basis E that embeds vectors in $\mathbb{F}_p^a$ to elements in $\mathbb{F}_{p^b}$ (where $b = O(a)$), and an ϵ-biased generator $G\colon S^s \to \mathbb{F}_{p^b}^k$, there is a $(\mathbb{F}_{p^b}, k, t)$-tensor IOP, with non-adaptive queries, for the indexed automata relation R_{R1CSA} that supports instances over $\mathbb{F}_p$ with $n_{\text{row}} = Tw$, $n_{\text{col}} = (T+1)w = a \cdot k^t$, $n_{\text{in}} = a \cdot \ell_{\text{in}} \cdot k^{t_{\text{in}}}$, and has the following parameters:*

- *soundness error is $\max(t\epsilon, \epsilon_{\mathrm{MLA}}, O(\epsilon_{\mathrm{TSP}}))$;*
- *round complexity is $O(\max(\mathsf{r}_{\mathrm{MLA}}, \mathsf{r}_{\mathrm{TSP}}))$;*
- *proof length is $O(n_{\text{row}}/a + \mathsf{l}_{\mathrm{MLA}} + \mathsf{l}_{\mathrm{TSP}})$ elements in $\mathbb{F}_{p^b}$;*
- *query complexity is $O(\mathsf{qc}_{\mathrm{MLA}} + \mathsf{qc}_{\mathrm{TSP}})$;*
- *the prover sends $O(\mathsf{c}_{\mathrm{MLA}} + \mathsf{c}_{\mathrm{TSP}})$ non-oracle messages over $\mathbb{F}_{p^b}$;*
- *the prover uses $O(M + n_{\text{row}})$ $\mathbb{F}_p$-operations and $O(t \cdot (\mathsf{o}_G + n_{\text{row}}/a))$ $\mathbb{F}_{p^b}$-operations in addition to the $O(\mathsf{tp}_{\mathrm{MLA}} + \mathsf{tp}_{\mathrm{TSP}})$ operations from the sub-protocols;*
- *the verifier uses $O(b^3)$ $\mathbb{F}_p$-operations and $O(t \cdot \mathsf{o}_G)$ $\mathbb{F}_{p^b}$-operations in addition to the $O(\mathsf{tv}_{\mathrm{MLA}} + \mathsf{tv}_{\mathrm{TSP}})$ operations from the sub-protocols;*
- *the verifier has randomness complexity $O(ts)$ over S in addition to the $O(\mathsf{rd}_{\mathrm{MLA}} + \mathsf{rd}_{\mathrm{TSP}})$ randomness for the sub-protocols.*

Here M denotes the number of non-zero entries in the instance matrices A, B, C as in Section 6.3, and due to the special structure of staircase matrices, we know $M = w^2 T$.

7.3 Cyclic-shift test

We construct a tensor IOP for the shift relation.

Definition 7.12. *The* **shift** *relation $R_{\circlearrowright}$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\bot, (\mathbb{F}, s, N), (a, b))$$

where $N = k^t$, $b \in \mathbb{F}^N$ is the cyclic shift *of $a \in \mathbb{F}^N$, which we denote $b = \mathrm{shift}(a)$, if for all $i \in [N - s]$, it holds that $a_i = b_{i+s}$, and for $i \geq N - s$, $a_i = 0$.*

Theorem 7.13. *For every finite field $\mathbb{F}$ and positive integers k, t, s with $s < k$, there is a $(\mathbb{F}, k, t)$-tensor IOP for the indexed relation $R_{\circlearrowright}$ that supports instances over $\mathbb{F}$ with $N = k^t$ and has the following parameters:*

- *soundness error is $O(tk/|\mathbb{F}|)$;*
- *round complexity is $O(1)$;*
- *proof length is $O(N)$ elements in $\mathbb{F}$;*
- *query complexity is $O(t)$;*
- *the verifier uses $O(tk)$ field operations;*
- *the verifier has randomness complexity $O(t)$.*

We define vector notation which allows us to describe the protocol more succinctly.

Definition 7.14. *For $\gamma_1, \dots, \gamma_t \in \mathbb{F}$, and $v, w \in \{0, 1, \dots, k-1\}$ with $v \leq w$, let $\Gamma_j := (1, \gamma_j, \dots, \gamma_j^{k-1}) \in \mathbb{F}^k$, and let $\Gamma_j^{(v,w)} := (0, \dots, 0, \gamma_j^v, \gamma_j^{v+1}, \dots, \gamma_j^w, 0, \dots, 0) \in \mathbb{F}^k$.*

Construction 7.15 (tensor IOP for cyclic shift test)**.** We construct an interactive oracle proof $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ with tensor queries for the indexed relation $R_{\circlearrowright}$. The indexer algorithm $\mathbf{I}$ is trivial. The prover $\mathbf{P}$ takes as input the instance $\mathbb{x} = (n_{\text{in}}, x)$, and witness $\mathbb{w} = w$, while the verifier $\mathbf{V}$ takes as input the instance $\mathbb{x}$.

The verifier makes $2t+1$ tensor queries to check the following expressions:

$$\gamma_1^s \langle \Gamma_1^{(0,k-1-s)} \otimes \bigotimes_{j=2}^{t} \Gamma_j, a \rangle = \langle \Gamma_1^{(s,k-1)} \otimes \bigotimes_{j=2}^{t} \Gamma_j, b \rangle$$

$$\gamma_2 \gamma_1^{s-k} \langle \Gamma_1^{(k-s,k-1)} \otimes \Gamma_2^{(0,k-2)} \otimes \bigotimes_{j=3}^{t} \Gamma_j, a \rangle = \langle \Gamma_1^{(0,s-1)} \otimes \Gamma_2^{(1,k-1)} \otimes \bigotimes_{j=3}^{t} \Gamma_j, b \rangle$$

$$\vdots$$

$$\gamma_r \gamma_{r-1}^{1-k} \cdots \gamma_2^{1-k} \gamma_1^{s-k} \langle \Gamma_1^{(k-s,k-1)} \otimes \bigotimes_{j=2}^{r-1} \Gamma_j^{(k-1,k-1)} \otimes \Gamma_r^{(0,k-2)} \otimes \bigotimes_{j=r+1}^{t} \Gamma_j, a \rangle$$

$$= \langle \Gamma_1^{(0,s-1)} \otimes \bigotimes_{j=2}^{r-1} \Gamma_j^{(0,0)} \otimes \Gamma_r^{(1,k-1)} \otimes \bigotimes_{j=r+1}^{t} \Gamma_j, b \rangle$$

$$\vdots$$

$$\gamma_t \gamma_{t-1}^{1-k} \cdots \gamma_2^{1-k} \gamma_1^{s-k} \langle \Gamma_1^{(k-s,k-1)} \otimes \bigotimes_{j=2}^{t-1} \Gamma_j^{(k-1,k-1)} \otimes \Gamma_r^{(0,k-2)}, a \rangle = \langle \Gamma_1^{(0,s-1)} \otimes \bigotimes_{j=2}^{t-1} \Gamma_j^{(0,0)} \otimes \Gamma_t^{(1,k-1)}, b \rangle$$

$$\langle \Gamma_1^{(k-s,k-1)} \otimes \bigotimes_{j=2}^{t} \Gamma_j^{(k-1,k-1)}, b \rangle = 0$$

Proof. We analyze the error and efficiency parameters of the above construction.

Completeness. Perfect completeness: Index the entries of $a, b \in \mathbb{F}^N$ using the k-ary representation $(i_t, \ldots, i_1) \in \{0, 1, \ldots, k-1\}^t$ (recall that $N = k^t$). Consider how the k-ary representation of i changes when adding s to get $i + s$. Since $s < k$, the only possible options are as follows.

- If $0 \leq i_1 \leq k-1-s$, then there are no carries when adding s to the k-ary representation $(i_t, \ldots, i_1)$. This means that $a_{i_t,\ldots,i_1}$ is equal to $b_{i_t,\ldots,i_1+s}$.
- If $k-s \leq i_1 \leq k-1$, and $0 \leq i_2 \leq k-2$, then there is one carry when adding s to the k-ary representation $(i_t, \ldots, i_1)$. This means that $a_{i_t,\ldots,i_1}$ is equal to $b_{i_t,\ldots,i_2+1,i_1+s-k}$.
- If $k-s \leq i_1 \leq k-1$, and $0 \leq i_2 \leq k-2$, then there are two carries when adding s to the k-ary representation $(i_t, \ldots, i_1)$. This means that $a_{i_t,\ldots,i_1}$ is equal to $b_{i_t,\ldots,i_3+1,0,i_1+s-k}$.

In the tensor queries in Construction 7.15, the $(i_t, \ldots, i_1)$-th entries of a and b either do not appear, or are multiplied by the monomial $\gamma_t^{i_t} \cdots \gamma_1^{i_1}$. Therefore, the number of carries gives the monomial which the $(i_t, \ldots, i_1)$-th term in a must be scaled by to give the corresponding term in b.

Soundness. Soundness error $O(tk/|\mathbb{F}|)$: Suppose that $a \neq \mathrm{shift}(b)$. This means that either $a_i \neq b$ for some $i \in [N-s]$, or $a_i \neq 0$ for some $i \geq N-s$. In the latter case, the final verification equation consists of a non-zero polynomial of degree at most kt evaluated at random points $\gamma_1, \ldots, \gamma_t \in \mathbb{F}$. The stated soundness error then follows from the Schwartz–Zippel lemma. In the former case, the number of carries in the k-ary representation of i when adding s to i determines which of the verification equations consists of a non-zero polynomial, and the stated soundness error follows in a similar fashion.

Efficiency parameters. All the efficiency parameters follow directly from the construction.

□

7.4 Biased generator consistency test

We explain how to check the correctness of the output for the subspace biased generator efficiently, with the assumption that we have a tensor-sum consistency test as described in Section 7.5.

Construction 7.16. Given an ϵ-biased generator $G\colon \mathbb{F}_{p^\lambda}^s \to \mathbb{F}_{p^\lambda}^{b'}$, let $G'(r_1, \ldots, r_t.r_{t+1}) := G(r_1) \otimes \cdots \otimes G(r_t) \otimes r_{t+1}$ for $r_1, \ldots, r_t \in \mathbb{F}_{p^\lambda}^s$ and $r_{t+1} \in \mathbb{F}_{p^\lambda}^{a/\lambda}$. Then, use Construction 5.5 to construct $G_{\mathsf{sub}}\colon \left(\mathbb{F}_{p^\lambda}^s\right)^t \to \mathbb{F}_p^{\lambda \times b'^t a}$ from G'.

By Lemma 5.2, G' is a $(t\epsilon + 1/p^\lambda)$-biased generator. Therefore, by Construction 5.5, $G_{\mathsf{sub}}\colon \left(\mathbb{F}_{p^\lambda}^s\right)^t \to \mathbb{F}_p^{\lambda \times \lambda b'^t}$ is a (p, λ)-subspace $(t\epsilon + 1/p^\lambda)$-biased generator.

Lemma 7.17. *Given an ϵ-biased generator $G\colon \mathbb{F}_{p^\lambda}^s \to \mathbb{F}_{p^\lambda}^{b'}$, let $G'\colon \left(\mathbb{F}_{p^\lambda}^s\right)^t \times \mathbb{F}_{p^\lambda}^{a/\lambda} \to \mathbb{F}_{p^\lambda}^{b'^t a/\lambda}$ and $G_{\mathsf{sub}}\colon \left(\mathbb{F}_{p^\lambda}^s\right)^t \times \mathbb{F}_{p^\lambda}^{a/\lambda} \to \mathbb{F}_p^{b'^t a}$ be the outputs from Construction 7.16. The matrix representation of the output of G_{sub} can be written as sums of tensor products.*

Proof. For all $r_1, \ldots, r_t \in \mathbb{F}_{p^\lambda}^s$ and $r_{t+1} \in \mathbb{F}_{p^\lambda}^{a/\lambda}$, we have $G'(r_1, \ldots, r_t.r_{t+1}) = G(r_1) \otimes \cdots \otimes G(r_t) \otimes r_{t+1}$. Indexing the entries of each $G(r_j)$ over $[b']$, r_{t+1} over $[a/\lambda]$, and $G'(r_1, \ldots, r_t)$ over $[b']^t$, we have

$$G'(r)_{i_1,\ldots,i_t,i_{t+1}} = G(r_1)_{i_1} \cdot \cdots \cdot G(r_t)_{i_t} \cdot (r_{t+1})_{i_{t+1}} \ . \tag{9}$$

Let $M^{(i,j)} \in \mathbb{F}_p^{\lambda \times \lambda}$ be the matrix representation of the j-th entry of $G(r_i) \in \mathbb{F}_{p^\lambda}^{b'}$ or $r_{t+1} \in \mathbb{F}_{p^\lambda}^{a/\lambda}$, as defined by Observation 5.4. Similarly, let $Y^{(i_1,\ldots,i_{t+1})} \in \mathbb{F}_p^{\lambda \times \lambda}$ be the matrix representation of the $(i_1, \ldots, i_{t+1})$-th entry of $G'(r) \in \mathbb{F}_{p^\lambda}^{b'^t a/\lambda}$.

Rewrite Equation (9) using the $Y^{(i_1,\ldots,i_{t+1})}$ and matrix multiplications of the $M^{(i,j)}$. For all $i_1, \ldots, i_t \in [b']$, $i_{t+1} \in [a/\lambda]$,

$$Y^{(i_1,\ldots,i_t)} = \prod_{k=1}^{t+1} M^{(k,i_k)} \ .$$

Writing out the matrix multiplication explicitly in terms of the entries of the $M^{(k,i_k)}$, the (m_0, m_{t+1})-th entry of $Y^{(i_1,\ldots,i_{t+1})}$ is equal to

$$Y^{(i_1,\ldots,i_{t+1})}_{m_0,m_{t+1}} = \left(\prod_{k=1}^{t+1} M^{(k,i_k)}\right)_{m_0,m_{t+1}} = \sum_{m_1,\ldots,m_t \in [\lambda]} \prod_{k=1}^{t} M^{(k,i_k)}_{m_{k-1},m_k} \ , \tag{10}$$

where the subscripts are row and column indices, respectively.

Now we explain how we to write the output of G_{sub} as a sum of tensor products. Define vectors $y^{(m_0,m_{t+1})} \in \mathbb{F}_p^{b'^t a}$ whose $(i_1, \ldots, i_{t+1})$-th entries are defined to be $y^{(m_0,m_{t+1})}_{i_1,\ldots,i_{t+1}} := Y^{(i_1,\ldots,i_{t+1})}_{m_0,m_{t+1}}$. Define vectors $v^{(k,m_{k-1},m_k)} \in \mathbb{F}_p^{b'}$ for $k \in [t]$ and $v^{(t+1,m_t,m_{t+1})} \in \mathbb{F}_p^{a/\lambda}$ whose i_k-th entries are defined to be $v^{(k,m_{k-1},m_k)}_{i_k} := M^{(k,i_k)}_{m_{k-1},m_k}$. Equation (10) can be rewritten as

$$y^{(m_0,m_{t+1})} = \sum_{m_1,\ldots,m_t \in [\lambda]} \bigotimes_{k=1}^{t+1} v^{(k,m_{k-1},m_k)} \ . \tag{11}$$

Equation (11) follows from Equation (10) by considering the $(i_1, \ldots, i_{t+1})$-th components of the equation.

Since the output of G_{sub} is Y can be described using the vectors $y^{(m_0,m_{t+1})} \in \mathbb{F}_p^{b'^t a}$ for $m_0, m_{t+1} \in [\lambda]$ (using the $Y^{(i_1,\ldots,i_{t+1})}$ as an intermediate step), we have succeeded in writing the output of G_{sub} as sums of tensor products. $\square$

Now we are ready to construct a tensor IOP for checking the output of subspace-biased generator using a tensor IOP for tensor-sum consistency in Section 7.5. Write ϵ_{TS} for the soundness error of the tensor-sum consistency protocol, r_{TS} for its round complexity, l_{TS} for its proof length, $\mathsf{qc}_{\mathrm{TS}}$ for its query complexity, c_{TS} for its communication complexity, $\mathsf{tp}_{\mathrm{TS}}$ for its prover arithmetic complexity, $\mathsf{tv}_{\mathrm{TS}}$ for its verifier arithmetic complexity, and $\mathsf{rd}_{\mathrm{TS}}$ for its randomness complexity. We also use $\mathsf{o}_{G_{\mathsf{sub}}}$ to denote the arithmetic complexity of computing G_{sub} and o_G to denote the arithmetic complexity of computing G.

Theorem 7.18. *For every prime power p, and ϵ-biased generator $G\colon \mathbb{F}_{p^\lambda}^s \to \mathbb{F}_{p^\lambda}^{b'}$, there is a $(\mathbb{F}_p, b', t)$-tensor IOP, with non-adaptive queries, for the output of a (p, λ)-subspace biased generator $G_{\mathsf{sub}}\colon \left(\mathbb{F}_{p^\lambda}^s\right)^t \to \mathbb{F}_p^{\lambda\times b'^t a}$ constructed using Construction 7.16 from G, and has the following parameters:*
- *soundness error is ϵ_{TS};*
- *round complexity is r_{TS};*
- *proof length is $O(\lambda^2 b'^t a)$ elements in $\mathbb{F}_p$ and l_{TS} from subprotocols;*
- *query complexity is $O(\lambda^2 \mathsf{qc}_{\mathrm{TS}})$;*
- *the prover uses $\mathsf{o}_{G_{\mathsf{sub}}}$ $\mathbb{F}_{p^\lambda}$-operations in addition to $\mathsf{tp}_{\mathrm{TS}}$ operations from the sub-protocols;*
- *the verifier uses $(t \cdot \mathsf{o}_G + a/\lambda)$ $\mathbb{F}_{p^\lambda}$-operations in addition to the $\mathsf{tv}_{\mathrm{TS}}$ operations from the sub-protocols;*
- *the verifier has randomness complexity $\mathsf{rd}_{\mathrm{TS}}$ from the sub-protocols.*

Construction 7.19 (tensor IOP for subspace biased generator)**.** We construct an interactive oracle proof $\mathsf{IOP} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ with tensor queries for checking the output of G_{sub} in Theorem 7.18. The indexer algorithm $\mathbf{I}$ is trivial.

The prover takes as input matrix $R = G_{\mathsf{sub}}(r) \in \mathbb{F}_p^{\lambda\times b'^t a}$,and seed $r = (r_1, \ldots, r_t, r_{t+1}) \in \left(\mathbb{F}_{p^\lambda}^s\right)^t \times \mathbb{F}_{p^\lambda}^{a/\lambda}$. The verifier takes as input $r_1, \ldots, r_t \in \mathbb{F}_{p^\lambda}^s$ and $r_{t+1} \in \mathbb{F}_{p^\lambda}^{a/\lambda}$.

- $\mathbf{P}$ sends oracle messages $y^{(m_0,m_{t+1})} \in \mathbb{F}_p^{b'^t a}$ defined as in Lemma 7.17;
- $\mathbf{V}$ computes $v^{(k,m,m')}$ for all $k \in [t+1], m, m' \in [\lambda]$ such that $v_{i_k}^{(k,m,m')} := M_{m,m'}^{(k,i_k)}$, where $M^{(k,i_k)}$ is the matrix representation of the i_k-th entry of $G(r_{i_k}) \in \mathbb{F}_{p^\lambda}^{b'}$ for $k \in [t]$, and the matrix representation of $(r_{t+1})_{i_{t+1}}$ when $k = t+1$;
- $\mathbf{P}$ and $\mathbf{V}$ engage in λ^2 tensor-sum consistency tests (defined in Section 7.5) to check if $y^{(m_0,m_{t+1})} = \sum_{m_1,\ldots,m_t} \bigotimes_{k=1}^{t+1} v^{(k,m_{k-1},m_k)}$.

Proof. We analyze the error and efficiency parameters of the above construction.

Completeness. If the output of G_{sub} is correctly computed, then all the tensor-sum consistency tests will pass according to Lemma 7.17.

Soundness. The soundness error is ϵ_{TS}: Suppose the output of G_{sub} is not correctly computed. Then at least one of the tensor-sum consistency passes incorrectly. The probability that this happens is ϵ_{TS}.

Efficiency parameters. The round complexity, query complexity, and verifier's randomness complexity are inherited from the tensor-sum consistency tests directly. The proof length is the length of the λ^2

oracle messages in $\mathbb{F}_p^{b'/t}$ and the proof length from the sub-protocols. The prover's and verifier's arithmetic complexities include the operations from the sub-protocols, and the operations to compute G_{sub} and G, respectively. $\square$

7.5 Tensor-sum consistency test

In Construction 7.9, $\mathbf{V}$ knows $r_{1,i}, \ldots, r_{t,i} \in \mathbb{F}^k$ for $i \in [\ell]$ and must check that $\mathbf{P}$ has sent $r \in \mathbb{F}^{k^t}$ that equals $\sum_{i=1}^{\ell} r_{1,i} \otimes \cdots \otimes r_{t,i}$. The test easily generalises to the case where the components $r_{1,i}, \ldots, r_{t,i}$ have different lengths.

Let $r_i := r_{1,i} \otimes \cdots \otimes r_{t,i}$. Using the same strategy as [**BootleCG20**], note that for every tensor $s = s_1 \otimes \cdots \otimes s_t \in \mathbb{F}^{k^t}$, we have $\langle r_i, s \rangle = \langle r_{1,i}, s_1 \rangle \cdots \langle r_{t,i}, s_t \rangle$, and if $r_i \neq r_{1,i} \otimes \cdots \otimes r_{t,i}$, then by the Schwartz–Zippel lemma, $\langle r_i, s \rangle \neq \langle r_{1,i}, s_1 \rangle \cdots \langle r_{t,i}, s_t \rangle$ except with probability at most $t/|\mathbb{F}|$ over the random choice of s. The same strategy can be applied to the sum of the r_i.

To check that $\sum_{i=1}^{\ell} r_{1,i} \otimes \cdots \otimes r_{t,i}$, the verifier samples random $s_1, \ldots, s_t \in \mathbb{F}^k$, and queries r at the tensor $s = s_1 \otimes \cdots \otimes s_t \in \mathbb{F}^{k^t}$. The verifier computes $\sum_{i=1}^{k} \langle r_{1,i}, s_1 \rangle \cdots \langle r_{t,i}, s_t \rangle$ in $O(\ell t k)$ arithmetic operations, and checks that the two values are equal. This gives rise to the following claim.

Claim 7.20. *In a $(\mathbb{F}, k, t)$-tensor IOP, where $\mathbf{V}$ has oracle access to $v \in \mathbb{F}^{k^t}$ and explicit input $v_{1,i}, \ldots, v_{t,i} \in \mathbb{F}^k$, $\mathbf{V}$ can check that $v = \sum_{i=1}^{\ell} v_{1,i} \otimes \cdots \otimes v_{t,i}$ with soundness error $t/|\mathbb{F}|$, using a single tensor query to v and $O(\ell t k)$ arithmetic operations.*

8 From point queries to tensor queries

We give a generalization of the compiler of [**BootleCG20**], which transforms a tensor-query IOP into a corresponding point-query IOP. That compiler relies on a proximity test for tensor codes which "folds" vectors via random linear combinations, using the fact that if any of the vectors is far from the code then the random linear combination is likely to be far from the code. Our generalization enables folding via any linear combination that satisfies the following distance preservation property.

Definition 8.1 ([**ChiesaKLM22**]). *We say that $G\colon \mathbb{F}^s \to \mathbb{F}^\ell$ is a (δ_0, ϵ)-proximity generator if for all n, all codes $\mathcal{C} \subseteq \mathbb{F}^n$, all $U \in \mathbb{F}^{\ell \times n}$ with (blockwise) relative distance $\delta_U = \delta(U, \mathcal{C}^\ell)$, and all $\Delta \in [0,1]$ such that $\Delta < \delta_0(\mathcal{C}, U)$, we have*

$$\Pr_{w \leftarrow \mathbb{F}^s}[\delta(G(w)^\intercal U, \mathcal{C}) \le \Delta] \le \epsilon(\Delta) \enspace .$$

The proximity generators used in prior work do not suffice for us. In [**BootleCL22**], $s = 1$ but $\epsilon = O(\ell \cdot n)$, which prevents the compiler from being used over fields with $o(n)$ elements; and in [**BootleCG20**], $\epsilon = O(1/|\mathbb{F}|)$ but $s = O(\ell)$, which is too inefficient for our purposes, as the verifier must send the entire seed for the proximity generator to the prover, and this incurs $O(k)$ communication costs in the consistency test. However, if we plug the proximity generator obtained in [**ChiesaKLM22**] (see Section 8.3) into our generalized compiler, then we obtain a compiler that with suitable efficiency over not-too-large fields.

Definition 8.2. *The indexed relation R_{cons} is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \left(\perp, (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t, \{q^{(s)}\}_s, \{v_s\}_s), c\right)$$

such that $c = \mathrm{Enc}_{\mathcal{C}^{\otimes t}}(f) \in \mathbb{F}^{\ell \cdot n^t}$ for some $f \in \mathbb{F}^{\ell \cdot k^t}$, for each $s \in [\mathsf{q}]$, $q^{(s)} = (q_0^{(s)}, \ldots, q_t^{(s)}) \in \mathbb{F}^\ell \times \left(\mathbb{F}^k\right)^t$, and for all $s \in [\mathsf{q}]$, $\langle \otimes_i q_i^{(s)}, f\rangle = v^{(s)}$.

Remark 8.3. For notational simplicity, when proving Lemma 8.4 which follows, we assume that the function δ_0 associated with proximity generators G and G' simply divides the relative distance δ_U of an interleaved codeword U by a constant factor κ. The analysis still applies to more general functions δ_0, resulting in iterated δ_0 functions in expressions for the soundness error.

Lemma 8.4. *Consider the following ingredients.*

- *A linear code $\mathcal{C}$ over $\mathbb{F}$ with rate $\rho = \frac{k}{n}$, relative distance $\delta = \frac{d}{n}$, and encoding arithmetic complexity $\Psi(k) \cdot k$.*
- *A function $G\colon \mathbb{F}^s \to \mathbb{F}^{(\mathsf{q}+1)k}$ such that for each $r \in [t-1]$, $\mathcal{C}^{\otimes r}$ has a (δ_0, ϵ) proximity gap with $\delta_0(\mathcal{C}^{\otimes r}, U) = \delta(U, \mathcal{C}^{\otimes r})/\kappa$, and G can be evaluated in o_G operations over $\mathbb{F}$.*
- *A function $G'\colon \mathbb{F}^{s'} \to \mathbb{F}^\ell$ with a (δ_0, ϵ) proximity gap with threshold $\delta_0(\mathcal{C}^{\otimes t}, U) = \delta(U, \mathcal{C}^{\otimes t})/\kappa$, and G' can be evaluated in $\mathsf{o}_G{}'$ operations over $\mathbb{F}$.*

There exists an non-adaptive interactive oracle proof of proximity $\mathsf{IOPP} = (\mathbf{P}, (\mathbf{V}_{\mathrm{q}}, \mathbf{V}_{\mathrm{d}}))$ with point queries for the relation $R_{\mathsf{cons}}(\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t, \{q^{(s)}\}_s, \{v_s\}_s)$ with the following parameters:

- *answer alphabet $\mathbb{F}^\ell$ for the witness, and $\mathbb{F}^k$ for the proof;*
- *soundness error $O\left(\sum_{r=1}^t \epsilon(\min\{\delta^{t-r}/4, \Delta_\otimes\})\right) + \left(1 - \kappa^{-2}\min\{\delta^t/4, \Delta_\otimes\}\right)^\lambda + \left(1 - \left(\delta^t - 2\min\{\delta^t/4, \Delta_\otimes\}\right)\right)^\lambda$;*

- *round complexity $O(t)$;*
- *proof length $O(\mathsf{q} \cdot n^{t-1})$;*
- *query complexity $O(\lambda)$ to the witness and $O(\lambda \cdot t \cdot \mathsf{q})$ to the proof;*
- *prover time $O(|\mathcal{C}| + \mathsf{q} \cdot t\ell k^t) + (t-1) \cdot \mathsf{o}_G + \mathsf{o}_G{}'$;*
- *verifier time $O(|\mathcal{C}| + \mathsf{q} \cdot (\ell + kt)) + (t-1) \cdot \mathsf{o}_G + \mathsf{o}_G{}'$;*
- *verifier randomness complexity $O(t \cdot \mathsf{q})$ elements in $\mathbb{F}$;*
- *$\mathbf{V}_{\mathrm{d}}$'s decision state $\sigma = (\mathbb{x}, r)$ where r is the verifier randomness;*
- *$\mathbf{V}_{\mathrm{d}}$ time $O(|\mathcal{C}| + \mathsf{q} \cdot (\ell + kt)) + (t-1) \cdot \mathsf{o}_G + \mathsf{o}_G{}'$.*

8.1 Proximity test

Definition 8.5. *The indexed relation $R_\otimes$ is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \Big(\bot, (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t), (c_0^{(0)}, \{c_1^{(s)}\}_s, \ldots, \{c_{t-1}^{(s)}\}_s)\Big)$$

such that $c_0^{(0)} \in (\mathcal{C}^{\otimes t})^\ell$ and, for all $r \in [t-1]$ and $s \in [\mathsf{q}]$, we have $c_r^{(s)} \in (\mathcal{C}^{\otimes t-r})^k$.

Definition 8.6. *Let $\mathbb{w} = (c_0^{(0)}, \{c_1^{(s)}\}_s, \ldots, \{c_{t-1}^{(s)}\}_s)$ be such that $c_0^{(0)} \in \mathbb{F}^{\ell \cdot n^t}$ and, for all $r \in [t-1]$ and $s \in [\mathsf{q}]$, we have $c_r^{(s)} \in \mathbb{F}^{k \cdot n^{t-r}}$. Given $(\mathbb{i}, \mathbb{x}) = (\bot, (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t))$, the $\Delta_\otimes$-distance of $\mathbb{w}$ to $R_{\mathsf{cons}}|_{(\mathbb{i},\mathbb{x})}$ is*

$$\Delta_\otimes\left(\mathbb{w}, R_{\mathsf{cons}}|_{(\mathbb{i},\mathbb{x})}\right) := \max\{\Delta_0, \Delta_1, \ldots, \Delta_{t-1}\} \quad \textit{where} \quad \begin{cases} \Delta_0 := \Delta(c_0^{(0)}, \mathcal{C}^{\otimes t}) \\ \forall\, r \in [t-1]\,,\ \Delta_r := \Delta(\{c_r^{(s)}\}_s, \mathcal{C}^{\otimes t-r}) \end{cases} .$$

Definition 8.7. *The* **folding** *of a function $c\colon [a] \times [b_1] \times \cdots \times [b_h] \to \mathbb{F}$ by a linear combination $\alpha\colon [a] \to \mathbb{F}$ is the function $\mathrm{Fold}(c; \alpha)\colon [b_1] \times \cdots \times [b_h] \to \mathbb{F}$ defined as follows:*

$$\mathrm{Fold}(c; \alpha) := \sum_{i \in [a]} \alpha(i) c(i, \cdot, \ldots, \cdot) \ . \tag{12}$$

Moreover, the folding of a set of functions $\{c_s\}_s$ by a set of corresponding linear combinations $\{\zeta_s\}_s$ is the function $\mathrm{Fold}(\{c_s\}_s; \{\zeta_s\}_s)\colon [b_1] \times \cdots \times [b_h] \to \mathbb{F}$ defined as follows:

$$\mathrm{Fold}(\{c_s\}_s; \{\zeta_s\}_s) := \sum_s \mathrm{Fold}(c_s; \zeta_s) \ .$$

More generally, for $r \in \{0, 1, \ldots, h\}$, we write Fold_r to indicate a folding operation that is applied to the r-th coordinate in the sum in Equation (12), as opposed to the 0-th coordinate.

Remark 8.8. For efficiency reasons, we assume that the honest prover $\mathbf{P}'$ also receives the *decodings* $(f_0^{(0)}, \{f_1^{(s)}\}_s, \ldots, \{f_{t-1}^{(s)}\}_s)$ of the witness elements $(c_0^{(0)}, \{c_1^{(s)}\}_s, \ldots, \{c_{t-1}^{(s)}\}_s)$ in the construction below.

Construction 8.9. We describe the construction of $\mathsf{IOPP} = (\mathbf{P}', \mathbf{V}')$. The prover $\mathbf{P}'$ takes as input an index $\mathbb{i} = \bot$, instance $\mathbb{x} = (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t)$, witness codewords $\mathbb{w} = (c_0^{(0)}, \{c_1^{(s)}\}_s, \ldots, \{c_{t-1}^{(s)}\}_s)$ and underlying messages $(f_0^{(0)}, \{f_1^{(s)}\}_s, \ldots, \{f_{t-1}^{(s)}\}_s)$, while the verifier $\mathbf{V}'$ takes as input the index $\mathbb{i}$ and the instance $\mathbb{x}$.

1. *Interactive phase.* For each round $r \in [t]$:
   - $\mathbf{V}'$ sends random challenge message $w \in \mathbb{F}^s$. (For $r = 1$, $\mathbf{V}'$ sends $w \in \mathbb{F}^{s'}$.)

- $\mathbf{P}'$ sends the proof message $c_r^{(0)} \in \mathbb{F}^{k \cdot n^{t-r}}$ computed as

$$f_r^{(0)} := \mathrm{Fold}_{r-1}(\{f_{r-1}^{(s)}\}_s; G(w)) \quad \text{and} \quad c_r^{(0)} := \mathrm{Enc}_{r+1,\dots,t}(f_r^{(0)}) \ .$$

(For $r = 1$, $\mathbf{P}'$ uses $\mathrm{Fold}_{r-1}(\{f_{r-1}^{(s)}\}_s; G'(w))$ instead.)

Note that when $r = t$ the expression $\mathrm{Enc}_{r+1,\dots,t}$ is degenerate and no encoding takes place.

2. *Query phase.* The verifier $\mathbf{V}'$ samples λ tuples of the form $(j_1, \dots, j_t) \leftarrow [n]^t$ and proceeds as follows for each tuple. For each $s \in \{0, 1, \dots, \mathsf{q}\}$, $\mathbf{V}'$ queries the function $c_r^{(s)} \colon [k] \times [n]^{t-r} \to \mathbb{F}$ at $(i_r, j_{r+1}, \dots, j_t)$ for each $i_r \in [k]$ and checks, for each $r \in [t]$, the following equation (replacing G with G' when $r = 1$):

$$\mathrm{Fold}_{r-1}(\{c_{r-1}^{(s)}\}_s; G(w))(j_r, \dots, j_t) = \mathrm{Enc}(c_r^{(0)})(j_r, \dots, j_t) \ . \tag{13}$$

Lemma 8.10. *The proximity test is sound with soundness error*

$$O\left(\sum_{r=1}^{t} \epsilon(\min\{\delta^{t-r}/4, \Delta_\otimes\})\right) + \left(1 - \kappa^{-2} \min\{\delta^{t-1}/4, \Delta_\otimes\}\right)^\lambda \ .$$

Proof sketch. If $(\mathbb{i}, \mathbb{x}) \notin L(R_\otimes)$, then the witness $\mathbb{w}$ and the oracle messages sent by a malicious prover $\mathbf{P}'$ define functions $c_r^{(s)}$ for $r \in [t]$ and $s \in [\mathsf{q}]$. If $\mathbf{P}'$ were honest, each oracle message $c_r^{(s)}$ would lie in $(\mathcal{C}^{\otimes t-r})^k$ and $c_0^{(0)}$ would lie in $(\mathcal{C}^{\otimes t})^\ell$.

As in [**BootleCG20**], the soundness analysis can be separated into two cases, each accounting for one of the terms in the soundness error. For each $r \in \{0, 1, \dots, t\}$, we define $\Delta_r^\dagger := \min\{\delta^{t-r}/4, \Delta_\otimes\}$.

- *Case 1:* One of the proximity generators fails (described as distortion in [**BootleCG20**]). This means that for some $r \in [t]$, we have $\Delta(\mathrm{Fold}(\{c_r^{(s)}\}_s; G(w)), \mathcal{C}^{\otimes t-r}) < \Delta_r^\dagger/\kappa$.

  Since G is a proximity generator, this occurs with probability at most $\epsilon(\min\{\Delta_r, \Delta_r^\dagger\})$. This replaces the use of a specific proximity generator from [**AmesHIV17**].

  Applying a union bound, the probability that any proximity generator fails is bounded above by

$$O\left(\sum_{r=1}^{t} \epsilon(\min\{\Delta_r, \Delta_r^\dagger\})\right) \ .$$

- *Case 2:* None of the proximity generators fail. At least one of the proof messages is far from being a codeword, or the proof messages are close to being codewords but the decoded messages from different rounds of the interactive phase are not consistent. Formally, for all $r \in \{0, 1, \dots, t-1\}$, we have $\Delta(\mathrm{Fold}(\{c_r^{(s)}\}_s; G(w)), \mathcal{C}^{\otimes t-r}) \geq \Delta_r^\dagger/\kappa$ but there exists $r \in \{0, 1, \dots, t-1\}$ such that one of the following conditions is satisfied:

  - $\Delta_r \geq \Delta_r^\dagger$;
  - $\Delta_r < \Delta_r^\dagger$ and $\mathrm{Fold}_r(\{\overline{c}_r^{(s)}\}_s; G(w)) \neq \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(0)})$.

  Here, $\overline{c}_r^{(s)}$ is the closest codeword to $c_r^{(s)}$. This closest codeword exists and is unique if the second condition is satisfied, since $\Delta_r < \Delta_r^\dagger$.

In this case, the analysis from [**BootleCG20**] holds, with $\Delta^{\dagger}_{r^*}$ replaced by $\Delta^{\dagger}_{r^*}/\kappa$ in [**BootleCG20**]. This gives a lower bound of $\Delta^{\dagger}_{r^*}/\kappa$ on the probability that verifier queries lie in "failure" and "error" sets. The analysis from [**BootleCG20**] then adapts to shows that the verifier will reject at least a $1/\kappa$ fraction of all queries in the failure or error sets, leading to a probability of $\min_r\{\Delta^{\dagger}_r\} = \min\{\delta^t/4, \Delta_{\otimes}\}$ that the verifier will reject when $\lambda = 1$. This leads to the stated soundness error.

□

8.2 Consistency checks on tensor codes

Construction 8.11. We describe the construction of $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}})$. The indexer $\hat{\mathbf{I}}$, given an index $\mathbb{i}$, runs $\mathbf{I}$ on $\mathbb{i}$ to produce $\Pi_0 \in \mathbb{F}^{\ell_0 \cdot k^t}$ indexed by $(i_0, i_1, \dots, i_t) \in [\ell_0] \times [k]^t$; then computes and outputs $\hat{\Pi}_0 := \mathrm{Enc}_{1,\dots,t}(\Pi_0) \in \mathbb{F}^{\ell_0 \cdot n^t}$.

The prover $\hat{\mathbf{P}}$ receives as input an instance $\mathbb{x}$ and witness $\mathbb{w}$, while the verifier $\hat{\mathbf{V}}$ receives as input the instance $\mathbb{x}$. The construction has two phases, a simulation phase and a consistency phase. We describe each in turn.

Simulation phase. For each $i \in [\mathsf{rc}]$, $\hat{\mathbf{P}}$ and $\hat{\mathbf{V}}$ simulate the i-th round of the interaction between $\mathbf{P}(\mathbb{x}, \mathbb{w})$ and $\mathbf{V}(\mathbb{x})$, as well as any tensor queries by $\mathbf{V}$ to the received proof strings.

1. *Prover messages.* $\hat{\mathbf{P}}$ receives from $\mathbf{P}$ a proof message $\Pi_i \in \mathbb{F}^{\ell_i \cdot k^t}$ indexed by $(i_0, i_1, \dots, i_t) \in [\ell_i] \times [k]^t$, computes a new proof message $\hat{\Pi}_i := \mathrm{Enc}_{t,\dots,t}(\Pi_i) \in \mathbb{F}^{\ell_i \cdot n^t}$, and sends $\hat{\Pi}_i$ to $\hat{\mathbf{V}}$. Also, $\hat{\mathbf{P}}$ forwards any non-oracle messages from $\mathbf{P}$ to $\mathbf{V}$ via $\hat{\mathbf{V}}$.

2. *Verifier messages.* $\hat{\mathbf{V}}$ receives challenge message ρ_i from $\mathbf{V}$ and forwards it to $\hat{\mathbf{P}}$, who forwards it to $\mathbf{P}$.

3. *Tensor queries.* If $\hat{\mathbf{V}}$ receives any tensor query (or queries) $q \in \mathcal{Q}_{\mathrm{tensor}}(\mathbb{F}, k, t)$ on $(\Pi_0, \Pi_1, \dots, \Pi_i)$ from $\mathbf{V}$, it sends the query q to $\hat{\mathbf{P}}$, who responds by computing $q(\mathbb{x}, \Pi_0, \Pi_1, \dots, \Pi_i)$ themselves and sending it to $\hat{\mathbf{V}}$ as a non-oracle message. Then $\hat{\mathbf{V}}$ forwards this (alleged) query answer to $\mathbf{V}$.

This completes the simulation of the tensor IOP. If at this point the tensor IOP verifier $\mathbf{V}$ rejects, then the IOP verifier $\hat{\mathbf{V}}$ rejects too. (There is no need to check if the IOP prover $\hat{\mathbf{P}}$ answered tensor queries honestly.)

Consistency phase. In this phase the IOP verifier $\hat{\mathbf{V}}$ checks that the IOP prover $\hat{\mathbf{P}}$ honestly answered the tensor queries of the tensor IOP verifier $\mathbf{V}$ in the simulation phase. Suppose that the tensor queries of $\mathbf{V}$ are given by $q^{(s)} = (q_0^{(s)}, q_1^{(s)}, \dots, q_t^{(s)})$ for each $s \in [\mathsf{q}]$. They are known to $\hat{\mathbf{P}}$ and $\hat{\mathbf{V}}$. They interact as follows.

1. *Send codewords.* The prover $\hat{\mathbf{P}}$ sends proof messages $\{c_r^{(1)}, \dots, c_r^{(\mathsf{q})} \in \mathbb{F}^{k \cdot n^{t-r}}\}_{r \in [t]}$ that are computed as described below.

   - First, define the functions

$$f_0^{(0)} := \mathrm{Stack}(\Pi_0, \dots, \Pi_{\mathsf{rc}}) \in \mathbb{F}^{\ell \cdot k^t} \quad \text{and} \quad c_0^{(0)}(i_0, j_1, \dots, j_t) := \mathrm{Stack}\left(\hat{\Pi}_0, \dots, \hat{\Pi}_{\mathsf{rc}}\right) \in \mathbb{F}^{\ell \cdot n^t} \ .$$

   Note that $c_0^{(0)} = \mathrm{Enc}_{1,\dots,t}(f_0^{(0)})$. Moreover, $\hat{\mathbf{P}}$ already knows the value of $f_0^{(0)}$ and of $c_0^{(0)}$ at every point, and $\hat{\mathbf{V}}$ has point-query access to every value of $c_0^{(0)}$ from the index or the messages sent during the simulation phase.

- Next, for each $r \in [t]$ and $s \in [\mathsf{q}]$, $\hat{\mathbf{P}}$ computes the following message and its encoding:

$$f_r^{(s)} := \mathrm{Fold}_{r-1}(f_{r-1}^{(s)}; q_{r-1}^{(s)}) \in \mathbb{F}^{k \cdot k^{t-r}} \quad \text{and} \quad c_r^{(s)} := \mathrm{Enc}_{r+1,\ldots,t}(f_r^{(s)}) \in \mathbb{F}^{k \cdot n^{t-r}} \enspace .$$

  For $r = 1$, $\mathrm{Fold}_{r-1}(f_{r-1}^{(0)}; q_{r-1}^{(s)})$ is used. When $r = t$, the expression $\mathrm{Enc}_{r+1,\ldots,t}$ is degenerate and no encoding takes place, and so $c_t^{(1)}, \ldots, c_t^{(\mathsf{q})}$ are vectors in $\mathbb{F}^k$.

2. *Proximity test.* The prover $\hat{\mathbf{P}}$ and verifier $\hat{\mathbf{V}}$ engage in the IOP of proximity $\mathsf{IOPP} = (\mathbf{P}', \mathbf{V}')$ for $R_\otimes(\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t)$ with index $\mathbb{i} = \bot$, instance $\mathbb{x} = (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t)$, and witness $\mathbb{w} = (c_0^{(0)}, \{c_1^{(s)}\}_s, \ldots, \{c_{t-1}^{(s)}\}_s)$ to show that $c_0^{(0)} \in (\mathcal{C}^{\otimes t})^\ell$ (or at least close) and that $c_r^{(s)} \in (\mathcal{C}^{\otimes t-r})^k$ (or at least close) for all $r \in [t]$ and $s \in [\mathsf{q}]$. (If $\mathbf{V}'$ rejects in this sub-protocol, then $\hat{\mathbf{V}}$ rejects.)

3. *Consistency checks.* The verifier $\hat{\mathbf{V}}$ samples λ tuples of the form $(j_1, \ldots, j_t) \leftarrow [n]^t$ and, for each tuple $(j_1, \ldots, j_t)$, proceeds as follows. For each $r \in [t]$, each $s \in [\mathsf{q}]$, and each $i_r \in [k]$, the verifier $\hat{\mathbf{V}}$ queries the function $c_r^{(s)} \colon [k] \times [n]^{t-r} \to \mathbb{F}$ at $(i_r, j_{r+1}, \ldots, j_t)$. Then, for each $r \in [t]$ and $s \in [\mathsf{q}]$, $\hat{\mathbf{V}}$ checks the following equation:

$$\mathrm{Fold}_{r-1}(c_{r-1}^{(s)}; q_{r-1}^{(s)})(j_r, \ldots, j_t) = \mathrm{Enc}_r(c_r^{(s)})(j_r, \ldots, j_t) \enspace .$$

   Finally, for each $s[\mathsf{q}]$, $\hat{\mathbf{V}}$ computes $\mathrm{Fold}_t(c_t^{(s)}; q_t^{(s)})$, and checks that it is equal to the answer to the s-th tensor query $q^{(s)} = (q_0^{(s)}, q_1^{(s)}, \ldots, q_t^{(s)})$ that was reported by $\hat{\mathbf{P}}$ in the simulation phase.

Lemma 8.12 (soundness). $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}})$ *has soundness error*

$$\epsilon'(\Delta_\otimes) + \left(1 - \left(\delta^t - 2\min\{\delta^t/4, \Delta_\otimes\}\right)\right)^\lambda \enspace .$$

Proof sketch. If $(\mathbb{i}, \mathbb{x}) \notin L(R)$ then the witness $\mathbb{w}$ and oracle messages sent by a malicious prover $\hat{\mathbf{P}}$ define a word $c_0^{(0)} \in \mathbb{F}^{\ell \cdot n^t}$ which, in an honest proof, would be ℓ interleaved $\mathcal{C}^{\otimes t}$-codewords, and word $c_r^{(s)} \in \mathbb{F}^{k \cdot n^{t-r}}$ for each $r \in [t]$ and $s \in [\mathsf{q}]$, where, in an honest proof, $c_r^{(s)}$ would be k interleaved $\mathcal{C}^{\otimes t-r}$-codewords.

We separate the soundness proof into different cases.

- *Case 1:* At least one of the proof messages is far from being a codeword. Formally, there exists $r \in \{0, 1, \ldots, t-1\}$ such that $\Delta_r \geq \delta^{t-r}/4$. By soundness of the proximity test, it follows that $(\mathbf{P}', \mathbf{V}')$, and hence $\hat{\mathbf{V}}$, will accept with probability at most $\epsilon'(\Delta_\otimes)$.

- *Case 2:* The proof messages are close to being codewords but the decodings of words $c_r^{(s)}$ from the consistency phase are not consistent across values of r. Formally, for all $r \in \{0, 1, \ldots, t-1\}$ we have $\Delta_r < \delta_0(\mathcal{C}^{\otimes t-r})$, but there exists $r \in \{0, 1, \ldots, t-1\}$ and $s \in [\mathsf{q}]$ such that $\mathrm{Fold}_r(\overline{c}_r^{(s)}; q_r^{(s)}) \neq \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(s)})$.

  In this case, the analysis in Case 2 of [**BootleCG20**] shows that $\mathrm{Fold}_r(\overline{c}_r^{(s)}; q_r^{(s)}) \neq \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(s)})$ are at least $\delta^{t-r} - 2\min\{\delta^t/4, \Delta_\otimes\}$ apart in relative distance, so that the verifier will reject with probability at least this value. This follows from replacing $\delta^{t-r}/4$ by $\min\{\delta^t/4, \Delta_\otimes\}$ in the inequalities.

- *Case 3:* Every message is close to being a codeword, and the decodings of words $c_r^{(s)}$ are all consistent over different values of r. Formally, for all $r \in \{0, 1, \ldots, t-1\}$, we have $\Delta_r < \delta_0(\mathcal{C}^{\otimes t-r})$ so in particular, for all $r \in \{0, 1, \ldots, t-1\}$ and $s \in [\mathsf{q}]$, the closest codewords $\overline{c}_r^{(s)}$ are well-defined. Then, the consistency checks are satisfied by the corrected words $\overline{c}_r^{(s)}$ i.e. for all $r \in \{0, 1, \ldots, t-1\}$ and $s \in [\mathsf{q}]$,

$$\mathrm{Fold}_r(\overline{c}_r^{(s)}; q_r^{(s)}) = \mathrm{Enc}_{r+1}(\overline{c}_{r+1}^{(s)}) \ .$$

Decoding the codewords $\overline{c}_r^{(s)}$ for each $r \in \{0, 1, \ldots, t-1\}$ and $s \in [\mathsf{q}]$, gives functions $f_{r+1}^{(s)}$ consisting of k vectors in $\mathbb{F}^{k^{t-r}}$ (or ℓ vectors for $r = 0$) satisfying $f_{r+1}^{(s)} = \mathrm{Fold}_r(f_r^{(s)}; q_r^{(s)})$. This implies that $\mathrm{Fold}_t(c_t^{(s)}; q_t^{(s)})$ is equal to $\langle \otimes_i q_i^{(s)}, f \rangle = v^{(s)}$.

□

Efficiency parameters. The round complexity of the consistency test is the same as in [**BootleCG20**], with $O(\mathsf{rc})$ rounds for the simulation phase and $O(t)$ rounds for the consistency phase, including the proximity test.

The proof messages and queries are identical to those in the consistency test of [**BootleCG20**] but are now measured in terms of two different alphabet sizes. In the consistency test, the prover sends a message array of size $\ell \cdot k^t$, parsed over the alphabet $\mathbb{F}^\ell$, and $O(\mathsf{q})$ messages for each of the array sizes k^t, $k^{t-1}, \ldots, k$, parsed over the alphabet $\mathbb{F}^k$. Each message array is queried at $O(\lambda)$ alphabet symbols.

Indexer time is the same as in [**BootleCG20**]. The prover and verifier time are the same except that the random linear combinations used in [**BootleCG20**] are now replaced by the output of proximity generators, which incur extra running time $(t-1) \cdot \mathsf{o}_G + \mathsf{o}_{G}'$.

Contributions to the verifier randomness complexity in [**BootleCG20**] were dominated by various random linear combinations of lengths ℓk and $(\mathsf{q}+1)k$. In this construction, this contribution is replaced by the seed length of the proximity generators used. The remaining randomness complexity is inherited from the tensor IOP.

8.3 The proximity generator that we use

We rely on [**ChiesaKLM22**] to show that a multilinear generator is a proximity generator.

Definition 8.13. *We say that a function $G \colon \mathbb{F}^s \to \mathbb{F}^{(s/k)^k}$ is a degree-k multilinear generator if for any input $\{x_{j,h}\}_{j\in[k],h\in[s/k]}$ the function outputs the string $\{y_{h_1,\ldots,h_k}\}_{h_1,\ldots,h_k\in[s/k]^k}$ where $y_{h_1,\ldots,h_k} = \prod_{j\in[k]} x_{j,h_j}$.*

Lemma 8.14 ([**ChiesaKLM22**])**.** *A degree-k multilinear generator $G \colon \mathbb{F}^s \to \mathbb{F}^{(s/k)^k}$ is a (δ_0, ϵ)-proximity generator where $\delta_0(\mathcal{C}, U) = \delta(U, \mathcal{C}^{(s/k)^k})/2^k$ and $\epsilon = \frac{k}{|\mathbb{F}|-1}$.*

Proof. We prove the statement by induction on the degree k.

Base case. When $k = 1$, the generator G simply outputs the uniformly random distribution over $\mathbb{F}^s$. Then by [**RothblumVW13**], for any $\Delta \le \delta(U, \mathcal{C}^{(s/k)^k})/2$

$$\Pr_{w \leftarrow \mathbb{F}^s}[\delta(w^\intercal U, \mathcal{C}) \le \Delta] \le \frac{1}{|\mathbb{F}|-1} \ .$$

Inductive step. Suppose the statement holds for multilinear generators of degree $k-1$. Then we observe that the degree-k generator $G_k \colon \mathbb{F}^s \to \mathbb{F}^{(s/k)^k}$ can be constructed from the degree-$(k-1)$ generator $G_{k-1} \colon \mathbb{F}^{s(1-1/k)} \to \mathbb{F}^{(s/k)^{k-1}}$ as follows. Let $x \in \mathbb{F}^{s/k}$ and $w \in \mathbb{F}^{s(1-1/k)}$

$$G_k(x, w) = x_1 \cdot G_{k-1}(w) \| x_2 \cdot G_{k-1}(w) \| \ldots \| x_{s/k} \cdot G_{k-1}(w)$$

For any matrix $U \in \mathbb{F}^{(s/k)^k \times n}$, let $\{U_h\}_{h \in [s/k]}$ be the $(h-1) \cdot (s/k)^{k-1} + 1$-th to $h \cdot (s/k)^{k-1}$-th rows of U, and let u_h be the random variable $G_{k-1}(w)^\intercal U_h$ where $w \leftarrow \mathbb{F}^{s(1-1/k)}$. Finally use $U_{(k)} \in \mathbb{F}^{(s/k) \times n}$ to denote the matrix whose h-th row is u_h. Then for any $\Delta \leq \delta(U, \mathcal{C}^{(s/k)^k})/2^k$, define the event E_k that $\delta(U_{(k)}, \mathcal{C}^{s/k}) \geq \delta(U, \mathcal{C}^{(s/k)^k})/2^{k-1}$. We then obtain

$$\begin{aligned}
&\Pr_{(x,w) \leftarrow \mathbb{F}^s} [\delta(G_k(x,w)^\intercal U, \mathcal{C}) \leq \Delta] \\
= &\Pr_{(x,w) \leftarrow \mathbb{F}^s} \left[\delta(x^\intercal U_{(k)}, \mathcal{C}) \leq \Delta\right] \\
\leq &\Pr_{w \leftarrow \mathbb{F}^s} [E_k] \cdot \Pr_{(x,w) \leftarrow \mathbb{F}^s} \left[\delta(x^\intercal U_k, \mathcal{C}) \leq \delta(U_{(k)}, \mathcal{C}^{(s/k)^k})/2 \mid E_k\right] + \Pr_{w \leftarrow \mathbb{F}^s} \left[\overline{E_k}\right] \quad .
\end{aligned}$$

Let i^* be the row in U with the maximum distance to $\mathcal{C}$, and U_{h^*} be the submatrix that contains row i^*. Then

$$\Pr_{w \leftarrow \mathbb{F}^s} [E_k] \geq \Pr_{w \leftarrow \mathbb{F}^s} \left[\delta(G_{k-1}(w)^\intercal U_{h^*}, \mathcal{C}) \geq \delta(U, \mathcal{C}^\ell)/2^{k-1}\right] \geq 1 - \frac{k-1}{|\mathbb{F}|-1}$$

The last inequality holds by observing that $\delta(U, \mathcal{C}^\ell) = \delta(U_{h^*}, \mathcal{C}^{(s/k)^{k-1}})$ and then applying the inductive hypothesis. Then plugging this bound back to the inequality we obtain that

$$\Pr_{(x,w) \leftarrow \mathbb{F}^s} [\delta(G_k(x,w)^\intercal U, \mathcal{C}) \leq \Delta] \leq \left(1 - \frac{k-1}{|\mathbb{F}|-1}\right) \cdot \frac{1}{|\mathbb{F}|-1} + \frac{k-1}{|\mathbb{F}|-1} \leq \frac{k}{|\mathbb{F}|-1} \quad .$$

Thus we complete the proof. □

As a corollary we obtain proximity generators with exponential stretch by setting $k = s/2$.

Corollary 8.15. *For any positive integer c and any m, there exists a (δ_0, ϵ)-proximity generator $G \colon \mathbb{F}^{cm^{1/c}} \to \mathbb{F}^m$, where $\delta_0(\mathcal{C}, U) = \delta(U, \mathcal{C}^\ell)/2^c$ and $\epsilon = \frac{c}{|\mathbb{F}|-1}$.*

9 Random access machines and precomputation

The definition of a random access machine below captures the usual notion of "word RAM", and is similar to definitions in [**AngluinV77**; **CookR72**].

Definition 9.1. *A* **random access machine** $M = (P, R, w, I)$ *consists of:*

- R *working registers, each storing a word in* $\{0,1\}^w$ *(and initially set to* 0^w*);*
- 2^w *memory registers, each storing a word in* $\{0,1\}^w$ *(and initially set to* 0^w*);*
- *a program* P *applying instructions in* I *to the working registers;*
- *a read-only input tape containing a list of words in* $\{0,1\}^w$*; and*
- *a write-only output tape, containing a list of words in* $\{0,1\}^w$*, with every word initially set to* 0^w*.*

Instructions in I *are functions taking a fixed number of working registers as input and writing the output to a single working register. In addition:*

- *memory instructions in* I *allow writing from a working register to a memory register, and vice versa;*
- *read instructions allow reading a word from the input tape to a working register (and move that tape's head forward);*
- *write instructions allow writing a word from a working register to the output tape (and move that tape's head forward); and*
- *a special* `Halt` *instruction halts the execution of the machine.*

Definition 9.2. *The* **time** T *of a random access machine is the number of instructions executed by the machine before halting. The* **space** S *is the maximum number of non-zero memory registers throughout the execution. The* **output** *consists of the contents of the output tape after the machine halts.*

We assume that the instruction set I consists of standard RISC instructions (jumps, shifts, bitwise boolean operators, certain integer arithmetic, and so on), all of which take $O(1)$ registers as input.

The lemma below provides a generic way to precompute a specific operation, by computing its evaluation table once and subsequently looking up the desired entries in the table as needed.

Lemma 9.3. *Let* (P_O, R, w, I) *be a machine that evaluates a function* $O\colon \{0,1\}^{k\times w_O} \to \{0,1\}^{w_O}$ *in time* T_O *and space* S_O*. Let* $M = (P, R, w, I \cup \{O\})$ *be a RAM with word length* $w \geq kw_O$ *that runs in time* T *and space* S*. There exists a machine* $M' = (P', R, w, I)$ *that produces the same output as* M *and runs in time* $T' := T + 2^{kw_O} \cdot T_O$ *and space* $S' := S + 2^{kw_O} + S_O$*.*

Proof sketch. The program P' uses the program P_O to evaluate O at every possible input (of which there are 2^{kw_O}) and stores the results in memory in a look-up table. This can be done in time $2^{kw_O} \cdot T_O$ and space $2^{kw_O} + S_O$. The program P' then runs the program P but replaces "O" instructions with memory instructions that read the relevant entries in the stored look-up table. □

Precomputation of field operations. We define an instruction that realizes field operations, and then apply precomputation to realize operations over an extension field in terms of operations over the base field.

The instruction $O_{\mathbb{F}}$ defined below can be used to add or multiply elements of the finite field $\mathbb{F}$. Note that $O_{\mathbb{F}}$ can be implemented using standard RISC instructions.

Definition 9.4. *Let* p *be a prime power,* $b \in \mathbb{N}$*, and* $\mathbb{F}$ *a finite field with* p^b *elements. The* **field operation** $O_{\mathbb{F}}$ *is an instruction, compatible with a RAM with word length* $w \geq b \log p$*, that works as follows.*

- *The first, second, and third registers are each parsed as* $\lfloor \frac{w}{b \log p} \rfloor$ *"slots", with each slot containing an element of* $\mathbb{F}$ *stored as* b *elements of* $\mathbb{F}_p$ *in some fixed basis.*

- *The fourth register is parsed as a single bit denoting either addition or multiplication.*

- *The fifth, sixth, and seventh register are each parsed as three bit-strings of length* $\log\lfloor \frac{w}{b \log p} \rfloor$*, each addressing a single slot of the first, second, and third registers.*

The instruction $O_{\mathbb{F}}$*: parses the addressed slots of the first and second registers as elements of* $\mathbb{F}$*; adds or multiplies them over* $\mathbb{F}$ *according to the bit in the fourth register; and writes the result to the addressed slot in the third register.*

We use precomputation (Lemma 9.3) to replace applications of $O_{\mathbb{F}_{p^b}}$ with applications of $O_{\mathbb{F}_p}$.

Lemma 9.5. *Let* $M = (P, R, w, I \cup \{O_{\mathbb{F}_{p^b}}\})$ *be a RAM with word length* $w \geq 2b \log p$ *that runs in time* T *and space* S*. There is a RAM* $M' = (P', R, w, I \cup \{O_{\mathbb{F}_p}\})$ *that runs in time* $T + O(b^2 \cdot p^{2b})$ *and space* $S + O(p^{2b})$*.*

Proof. We discuss how to realize additions and then multiplications.

- *Addition.* An addition over $\mathbb{F}_{p^b}$ is straightforwardly realized via b additions over $\mathbb{F}_p$ (no precomputed lookup table is needed), since elements of $\mathbb{F}_{p^b}$ are stored in a basis representation over $\mathbb{F}_p$ for $O_{\mathbb{F}_p}$.

- *Multiplication.* We apply Lemma 9.3. The multiplication lookup table for $\mathbb{F}_{p^b}$ has size p^{2b}. The cost of changing the basis representation of an element of $\mathbb{F}_{p^b}$ into the standard basis over $\mathbb{F}_p$ is $O(b^2)$ operations in $\mathbb{F}_p$ (using a stored change-of-basis matrix in $\mathbb{F}_p^{b \times b}$). Then, the cost of multiplying two elements of $\mathbb{F}_{p^b}$ in the standard basis over $\mathbb{F}_p$ is $O(b^2)$ operations using a schoolbook multiplication algorithm. Finally, converting the resulting $\mathbb{F}_{p^b}$ element back into the desired basis over $\mathbb{F}_p$ costs $O(b^2)$ multiplications, again using a stored change-of-basis matrix.

□

10 Proof of main theorems

In this section, we explain how to construct linear-time point-query IOPs for R1CS and for automata computations over any finite field.

Theorem 10.1 (main)**.** *For every prime power p, there is a public-coin IOP (with point queries) for the indexed relation R_{R1CS} that supports instances over $\mathbb{F}_p$ with $N \times N$ matrices, each containing $M = \Omega(N)$ non-zero entries, that has the following efficiency parameters:*

- *soundness error is $O(1)$;*
- *round complexity is $O(\log N)$;*
- *answer alphabet is $\mathbb{F}_p$;*
- *proof length is $O(M + N)$;*
- *query complexity is $O(\log^2 N / \log p)$;*
- *the prover is a RAM with word length $w = \Theta(\log N)$, $R = O(1)$ registers that runs in $O(M + N)$ time and $O(N)$ space;*
- *the verifier is a RAM with word length $w = \Theta(\log N)$, $R = O(1)$ registers that runs in $O(M + N)$ time and $O(N)$ space.*

Theorem 10.2 (main)**.** *For every prime power p, and constant $t \in \mathbb{N}$, there is a public-coin IOP (with point queries) for the indexed automata relation R_{R1CSA} that supports instances over $\mathbb{F}_p$ with computation width w and computation time T, that has the following efficiency parameters:*

- *soundness error is $O(1)$;*
- *round complexity is $O(\log(wT))$;*
- *answer alphabet is $\mathbb{F}_p$;*
- *proof length is $O(wT)$;*
- *query complexity is $O(\log^2(wT) / \log p)$;*
- *the prover is a RAM with word length $w = \Theta(\log wT)$, $R = O(1)$ registers that runs in $O(w^2 T)$ time and $O(w^2 T)$ space;*
- *the verifier is a RAM with word length $w = \Theta(\log N)$, $R = O(1)$ registers that runs in $O((wT)^{1/t})$ time and $O((wT)^{1/t})$ space.*

We obtain these IOPs over $\mathbb{F}_p$ in four steps. Initially, we construct all protocols over an extension field $\mathbb{F}_{p^b}$ of $\mathbb{F}_p$. In Section 10.1, we construct a robust IOPP for checking consistency between encoded proof messages and answers to tensor queries. In Section 10.2 we use proof composition to improve the query complexity of the IOPP. In Section 10.3, we use the IOPP to compile the tensor-query IOPs for R_{R1CS} (Theorem 6.2) and R_{R1CSA} (Theorem 7.2) into point-query IOPs over $\mathbb{F}_{p^b}$. In Section 10.4, we use the preprocessing techniques of Section 9 to convert them into point-query IOPs over $\mathbb{F}_p$. The first three steps in the proof are direct adaptations of results in [**BootleCL22**], so we provide proof sketches (and more details can be found in [**BootleCL22**]).

10.1 Step 1: robustification

Corollary 10.3. *Let $\mathcal{C} \colon \mathbb{F}^k \to \mathbb{F}^{O(k)}$ be a linear code with constant rate, constant relative distance, and linear-time encoding, membership-deciding, and (error-free) decoding. For every $c \in \mathbb{N}$ the relation R_{cons} has a non-adaptive point-query IOPP $(\mathbf{P}_{\mathrm{r}}, (\mathbf{V}_{\mathrm{qr}}, \mathbf{V}_{\mathrm{dr}}))$ with:*

- *soundness error $O(t/|\mathbb{F}|) + O(1)$ with robustness $\Theta(1/(\ell + t))$;*

- *round complexity $O(t)$;*
- *answer alphabet $\mathbb{F}$;*
- *proof length $O(\mathsf{q} \cdot \ell k^t)$;*
- *query complexity $O(\mathsf{q}k \cdot (\ell + t))$;*
- *prover time $O(\mathsf{q} \cdot t\ell k^t)$;*
- *verifier time $O(\mathsf{q}k \cdot (\ell + t))$;*
- *verifier randomness complexity $O(\mathsf{q}k^{1/c}(\ell^{1/c} + t))$;*
- $\mathbf{V}_{\mathrm{dr}}$ *time $O(\mathsf{q}k \cdot (\ell + t))$.*

The notion of robustness is explained in [**BootleCL22**].

Proof sketch. The proof of Corollary 10.3 uses Lemma 8.4 to construct an IOPP for the new relation R'_{cons}, before converting this into a robust IOPP for R_{cons}.

Definition 10.4. *The indexed relation R'_{cons} is the set of tuples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \left(\perp, (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t, \{q^{(s)}\}_s, \{v_s\}_s), c\right)$$

such that $c = \mathrm{Enc}_{\mathcal{C}^{\otimes(t-1)}}(f) \in \mathbb{F}^{\ell k \cdot n^{t-1}}$ for some $f \in \mathbb{F}^{\ell \cdot k^t}$, for each $s \in [\mathsf{q}]$, $q^{(s)} = (q_0^{(s)}, \ldots, q_t^{(s)}) \in \mathbb{F}^\ell \times \left(\mathbb{F}^k\right)^t$, and for all $s \in [\mathsf{q}]$, $\langle q^{(s)}, f\rangle = v^{(s)}$.

First, instantiate the IOPP for R_{cons} in Lemma 8.4 using the proximity generator from Corollary 8.15, but parsing the message $f_0^{(0)}$ as a t-dimensional array of size $\ell k \cdot k^{t-1}$ rather than a $(t+1)$-dimensional array of size $\ell \cdot k^t$. The first components $q_0^{(s)} \in \mathbb{F}^\ell$ and $q_1^{(s)} \in \mathbb{F}^k$ of each tensor query are grouped together as a single component $q_0^{(s)} \otimes q_1^{(s)} \in \mathbb{F}^{\ell k}$, and the first "folding" step applies $q_0^{(s)} \otimes q_1^{(s)}$ to $f_0^{(0)}$. In Lemma 8.4, set the constant $\kappa = 2^c$. Use the proximity generators from Corollary 8.15 with seed lengths $ck^{1/c}$ and $c(\ell k)^{1/c}$. This gives an IOPP for R'_{cons} with the stated soundness error and verifier randomness complexity.

Robustify this IOPP (using the transformation in [**BootleCL22**]) by encoding each witness symbol and each proof symbol using the error-correcting code $\mathcal{C}$. In Lemma 8.4, the witness has alphabet $\mathbb{F}^{\ell k}$, and will be encoded in ℓ parts by applying $\mathcal{C}$ to blocks of length k. This converts a witness for the relation R'_{cons} into a witness for the relation R_{cons}. Thus the new witness can be viewed as ℓ codewords in $\mathcal{C}^{\otimes t}$, or equivalently $O(\ell k)$ codewords in $\mathcal{C}^{\otimes(t-1)}$. Proof elements have alphabet $\mathbb{F}^k$ and will be encoded using $\mathcal{C}$.

□

10.2 Step 2: composition

Lemma 10.5 ([**Mie09**; **RonZewiR19**]). *Fix functions $T(n) = \Omega(n)$, $\delta(n) > 0$, and $\epsilon(n) > 0$. Every relation R in $\mathsf{NTIME}(T)$ has a PCPP for R with: answer alphabet $\{0,1\}$; proof length $T(n) \cdot \mathsf{polylog}(T(n))$; query complexity $O(\log(1/\epsilon(n)) \cdot \log(1/\delta(n))/\delta(n))$; soundness error $\epsilon(n)$ with proximity parameter $\delta(n)$; prover time $\mathsf{poly}(T(n))$; verifier time $\mathsf{poly}(n, \log T(n))$.*

Lemma 10.6. *Let $\epsilon \in (0,1)$ be any constant. For instances of R_{cons} in which each query $q^{(s)}$ is described using a seed of length at most $O(t \cdot k)$, there is a point-query IOPP $(\mathbf{P}, \mathbf{V})$ for R_{cons} with:*

- *soundness error $O(t/|\mathbb{F}|) + O(1) + \epsilon$ with proximity parameter $\Theta(1/t)$;*
- *round complexity $O(t)$;*
- *answer alphabet $\mathbb{F}$;*
- *proof length $O(\mathsf{q} \cdot \ell k^t)$;*

- *query complexity* $O_\epsilon(t \log t)$*;*
- *prover time* $\mathsf{poly}(\mathsf{q}k \cdot (\ell + t))$ *operations in* $\mathbb{F}$*;*
- *verifier time* $\mathsf{poly}(\mathsf{q}k \cdot (\ell + t))$ *operations in* $\mathbb{F}$.

Proof sketch. Apply proof composition (using the transformation in [**BootleCL22**]) to the robust outer IOP provided by Corollary 10.3 with a properly chosen parameter c and the inner IOP of proximity provided by Lemma 10.5.

The inner relation $R(\mathbf{V}_{\mathrm{dr}})$ has instance $(\mathbb{x}, r) := (\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t, \{q^{(s)}\}_s, \{v_s\}_s, r)$, where r has size $O(\mathsf{q} \cdot k^{1/c}(\ell^{1/c} + t))$, and by assumption, the query set has a description of size $O(\mathsf{q} \cdot t \log k)$. This means that the instance size is $O(\mathsf{q} \cdot t \log k)$. The witness size of $R(\mathbf{V}_{\mathrm{dr}})$ is equal to the query complexity of the robust outer IOP and has size $O(\mathsf{q}k \cdot (\ell + t))$. The outer verifier's decision time is $T(|\mathbb{x}|) = O(\mathsf{q}k \cdot (\ell + t))$. Choosing a c large enough and substituting these values into Lemma 10.5 gives the proof length, prover time and verifier time written in the theorem.

The round complexity $O(t)$ is inherited from the robust outer IOPP, while the query complexity $O_\epsilon(t \log t)$ is obtained from the applying Lemma 10.5 with the proximity parameter $\delta = \Theta(1/t)$. $\square$

10.3 Step 3: tensor queries to point queries

We restate [**BootleCL22**], removing details related to zero-knowledge and error-correcting codes.

Lemma 10.7 ([**BootleCL22**])**.** *There exists an explicit polynomial-time transformation T that satisfies the following. The inputs to the transformation are as follows:*

- *A t-linear tensor code $\mathcal{C}^{\otimes t}$ over $\mathbb{F}$ with encoding function $\mathrm{Enc}_{\mathcal{C}^{\otimes t}}$, with rate $\rho = \frac{k^t}{n^t}$, relative distance $\delta = \frac{d^t}{n^t}$, encoding arithmetic complexity $\Psi(k) \cdot k^t$.*

- *An interactive oracle proof* $\mathsf{IOP} = (\mathbf{P}, \mathbf{V})$ *with queries in $\mathcal{Q}_{\mathrm{tensor}}(\mathbb{F}, k, t)$ for an indexed relation R with: soundness error ϵ; round complexity* rc*; proof length* $\mathsf{l} = \mathsf{li} + \mathsf{lp} = \ell k^t$*; query complexity* q*; arithmetic complexity* ti *for the indexer; arithmetic complexity* tp *for the prover; arithmetic complexity* tv *for the verifier.*

- *An interactive oracle proof of proximity* $\mathsf{IOPP} = (\mathbf{P}', \mathbf{V}')$ *with queries in $\mathcal{Q}_{\mathrm{point}}$ for the indexed relation $R_{\mathsf{cons}}(\mathbb{F}, \mathcal{C}, \ell, \mathsf{q}, t)$ with soundness error ϵ'; round complexity* rc'*; proof length* l'*; input query complexity* $\mathsf{q}'_{\mathbb{x}}$*, the number of verifier queries to the input oracle; proof query complexity* q'_{π}*, the number of verifier queries to the* IOPP *proof oracles; arithmetic complexity* tp' *for the prover; arithmetic complexity* tv' *for the verifier.*

The output of the transformation $(\hat{\mathbf{I}}, \hat{\mathbf{P}}, \hat{\mathbf{V}}) := T(\mathcal{C}, \mathsf{IOP}, \mathsf{IOPP})$ *is an interactive oracle proof with queries in $\mathcal{Q}_{\mathrm{point}}$ for the indexed relation R with the following parameters:*

- *soundness error* $\max(\epsilon, (1 - 1/|\mathbb{F}|)\epsilon'(1/4) + 1/|\mathbb{F}|)$*;*
- *round complexity* $\mathsf{rc} + \mathsf{rc}' + 1$*;*
- *proof length* $O\left(\frac{n^t}{k^t} \cdot \mathsf{l}\right) + \mathsf{l}'$*;*
- *query complexity* $2\mathsf{q}'_{\mathbb{x}} + \mathsf{q}'_{\pi}$*, where* $2\mathsf{q}'_{\mathbb{x}}$ *queries are to the* IOP *proof oracles, and* q'_{π} *queries are to the* IOPP *proof oracles;*
- *indexer time* $\mathsf{ti} + \mathsf{ti}'$*;*
- *prover time* $\mathsf{tp} + \mathsf{tp}'$*; and*
- *verifier time* $\mathsf{tv} + \mathsf{tv}'$.

Lemma 10.8. *There is a constant η such that, for every prime power p, and every $a \in \Omega(\log \log N)$, there is a point-query IOP, with non-adaptive queries, for the indexed relation R_{R1CS} that supports instances over $\mathbb{F}_p$ with $N = a \cdot k^t$ and $M = a \cdot \ell \cdot k^t$ and has the following parameters:*
- *answer alphabet is $\mathbb{F} := \mathbb{F}_{p^{\eta a}}$;*
- *soundness error is $O(t/|\mathbb{F}|) + O(1) + \epsilon$;*
- *round complexity is $O(t + \log(N/a))$;*
- *proof length is $O(N/a)$ elements in $\mathbb{F}$;*
- *query complexity $O_\epsilon(t \log t)$;*
- *the prover sends $O(\log(N/a))$ non-oracle messages over $\mathbb{F}$;*
- *the prover uses $O(M + N)$ $\mathbb{F}_p$-operations and $O(tN/a) + \mathsf{poly}(\mathsf{q}k \cdot (\ell + t))$ $\mathbb{F}$-operations*
- *the verifier uses $O(M + N)$ $\mathbb{F}_p$-operations and $O(tk) + \mathsf{poly}(\mathsf{q}k \cdot (\ell + t))$ $\mathbb{F}$-operations.*

Proof sketch. Set $N = a \cdot k^t$ and $M = a \cdot \ell \cdot k^t$ for some $k, t \in \mathbb{N}$. Theorem 6.2 gives a $(\mathbb{F}, k, t)$-tensor IOP for R_{R1CS} over $\mathbb{F}_p$, which has query complexity $\mathsf{q} = O(1)$, and for which each query can be described by a short seed of size $O(tk)$. Applying Lemma 10.7 with Theorem 6.2 as the input tensor IOP, Lemma 10.6 as the input proximity test, using Spielman codes over $\mathbb{F}$ (for which δ, Ψ and ρ are all constant) yields the desired complexity parameters. □

Lemma 10.9. *There is a constant η such that, for every prime power p, and every $a \in \Omega(\log \log wT)$, there is a point-query IOP, with non-adaptive queries, for the indexed relation R_{R1CSA} that supports instances over $\mathbb{F}_p$ with $w(T + 1) = a \cdot k^t$ and has the following parameters:*
- *answer alphabet is $\mathbb{F} := \mathbb{F}_{p^{\eta a}}$;*
- *soundness error is $O(t/|\mathbb{F}|) + O(1) + \epsilon$;*
- *round complexity is $O(t + \log(wT/a))$;*
- *proof length is $O(wT/a)$ elements in $\mathbb{F}$;*
- *query complexity $O_\epsilon(t \log t)$;*
- *the prover sends $O(\log(wT/a))$ non-oracle messages over $\mathbb{F}$;*
- *the prover uses $O(w^2 T)$ $\mathbb{F}_p$-operations and $O(twT/a) + \mathsf{poly}(\mathsf{q}k \cdot (\ell + t))$ $\mathbb{F}$-operations*
- *the verifier uses $O(a^3) + O(t(wT)^{1/t}) + \mathsf{poly}(\mathsf{q}k \cdot (\ell + t))$ $\mathbb{F}_p$-operations and $O(tk)$ $\mathbb{F}$-operations.*

Proof sketch. Set $w(T + 1) = a \cdot k^t$ for some $k, t \in \mathbb{N}$. Theorem 7.2 gives a $(\mathbb{F}, k, t)$-tensor IOP for R_{R1CSA} over $\mathbb{F}_p$, which has query complexity $\mathsf{q} = O(1)$, and for which each query can be described by a short seed of size $O(tk)$. Applying Lemma 10.7 with Theorem 7.2 as the input tensor IOP, Lemma 10.6 as the input proximity test, using Spielman codes over $\mathbb{F}$ (for which δ, Ψ and ρ are all constant) yields the desired complexity parameters. Since $\lambda = O(1)$, all $\mathbb{F}_{p^\lambda}$ operations in Theorem 7.2 are counted as $O(1)$ $\mathbb{F}_p$ operations. □

10.4 Step 4: Precomputing expensive operations

We complete the proofs of Theorem 10.1 and Theorem 10.2 by applying Lemma 9.5 to Lemma 10.8 and Lemma 10.9 in order to precompute expensive operations over $\mathbb{F}$.

Precomputing for Theorem 10.1. We justify each parameter of Theorem 10.1 in turn. Set the output parameter for the RMFE to be $a := \frac{1}{2\eta} \log_p N - \frac{1}{\eta} \log_p \log_p N$.

- The answer alphabet $\mathbb{F}$ from Lemma 10.8 becomes $\mathbb{F}_p$, simply by parsing each element of $\mathbb{F} = \mathbb{F}_{p^{\eta a}}$ as a vector of ηa elements of $\mathbb{F}_p$.

- The soundness error becomes $O(t/p^{\eta a}) + O(1) + \epsilon$. Since $a \in \Omega(\log\log N)$, and ϵ can be an arbitrarily small constant in $(0, 1)$, the soundness error is $O(1)$.
- The round complexity is $O(t + \log(N/a))$ which is $O(\log N)$.
- The proof length becomes $O(N)$ elements in $\mathbb{F}_p$, by parsing each element of $\mathbb{F} = \mathbb{F}_{p^{\eta a}}$ as a vector of ηa elements of $\mathbb{F}_p$ as before.
- The query complexity becomes $O_\epsilon(t\eta a \log t)$ elements of $\mathbb{F}_p$.
- The prover sends $O(\eta a \log(N/a))$ non-oracle messages over $\mathbb{F}_p$.
- Consider the prover as a RAM with word length $w = \Theta(\log N)$, which is sufficient to model calculations over $\mathbb{F}_p$ and $\mathbb{F}_{p^{\eta a}}$. The prover uses $O(M + N)$ $\mathbb{F}_p$-operations from $\mathbb{F}_p$ operations in Lemma 10.8, and $O(tN) + a \cdot \mathsf{poly}(\mathsf{q}k \cdot (\ell + t))$ $\mathbb{F}_p$-operations from looking up the results of $\mathbb{F}$ operations in Lemma 10.8. In addition, the prover incurs an extra cost of $O(a^2 p^{2\eta a})$ operations in $\mathbb{F}_p$ for precomputing the multiplication table of $\mathbb{F}$. By choice of η, this requires $O(N)$ operations and $O(N)$ memory.
- Consider the verifier as a RAM with word length $w = \Theta(\log N)$, which is sufficient to model calculations over $\mathbb{F}_p$ and $\mathbb{F}_{p^{\eta a}}$. The verifier uses $O(M + N)$ $\mathbb{F}_p$-operations from $\mathbb{F}_p$ operations in Lemma 10.8, and $O(tka) + \eta a \cdot \mathsf{poly}(\mathsf{q}k \cdot (\ell + t))$ $\mathbb{F}_p$-operations from looking up the results of $\mathbb{F}$ operations in Lemma 10.8. In addition, the verifier incurs an extra cost of $O(a^2 p^{2\eta a})$ operations in $\mathbb{F}_p$ for precomputing the multiplication table of $\mathbb{F}$. As before, this requires $O(N)$ operations and $O(N)$ memory.

Precomputation for Theorem 10.2. Setting $a := \frac{1}{2\eta} \log_p wT - \frac{1}{\eta} \log_p \log_p wT$ as in Theorem 10.1, the answer alphabet, soundness error, round complexity, proof length, query complexity, number of non-oracle messages and prover complexity in Theorem 10.2 follow similarly to Theorem 10.1.

We justify the verifier complexity. Consider the verifier as a RAM with word length $w = \Theta(\log N)$, which is sufficient to model calculations over $\mathbb{F}_p$ and $\mathbb{F}_{p^{\eta a}}$. The verifier uses $O(a^3) + O(t(wT)^{1/t}) + \mathsf{poly}(\mathsf{q}k \cdot (\ell + t))$ $\mathbb{F}_p$-operations and $O(tk)$ $\mathbb{F}$-operations in Lemma 10.9. Since the verifier for Lemma 10.9 uses only a sublinear number of operations, there is no need for the verifier to precompute the multiplication table of $\mathbb{F}$. As such, the verifier simply uses $O(a^3) + O(t(wT)^{1/t}) + O(tk) \cdot \mathsf{polylog}(tk) + \mathsf{poly}(\mathsf{q}k \cdot (\ell + t))$ time and memory, where the $O(tk) \cdot \mathsf{polylog}(tk)$ cost comes from using multiplication algorithms such as the Schönhage-Strassen algorithm to compute $O(tk)$ $\mathbb{F}$-operations using operations in $\mathbb{F}_p$. Taking Lemma 10.9 with a larger value of t than given in Theorem 10.2, we can subsume this polylogarithmic factor inside the $O(t(wT)^{1/t})$ term, which yields the result.

Acknowledgments

We thank Noga Ron-Zewi and Ron Rothblum for their detailed comments which helped us to improve the paper. This research was supported in part by: the Berkeley Haas Blockchain Initiative and a donation from the Ethereum Foundation.