FRACTAL:
Post-Quantum and Transparent Recursive Proofs from Holography

Alessandro Chiesa
alexch@berkeley.edu
UC Berkeley

Dev Ojha
dojha@berkeley.edu
UC Berkeley

Nicholas Spooner
nick.spooner@berkeley.edu
UC Berkeley

October 25, 2019

Abstract

We present a new methodology to efficiently realize recursive composition of succinct non-interactive arguments of knowledge (SNARKs). Prior to this work, the only known methodology relied on pairing-based SNARKs instantiated on cycles of pairing-friendly elliptic curves, an expensive algebraic object. Our methodology does not rely on any special algebraic objects and, moreover, achieves new desirable properties: it is *post-quantum* and it is *transparent* (the setup is public coin).

We exploit the fact that recursive composition is simpler for SNARKs with *preprocessing*, and the core of our work is obtaining a preprocessing zkSNARK for rank-1 constraint satisfiability (R1CS) that is post-quantum and transparent. We obtain this latter by establishing a connection between holography and preprocessing in the random oracle model, and then constructing a holographic proof for R1CS.

We experimentally validate our methodology, demonstrating feasibility in practice.

Keywords: succinct arguments; holographic proofs; recursive proof composition; post-quantum cryptography

Contents

1 Introduction

Succinct non-interactive arguments (SNARGs) are cryptographic proofs for non-deterministic languages that are small and easy to verify. In the last few years, researchers from across multiple communities have investigated many aspects of SNARGs, including constructions under different cryptographic assumptions, improvements in asymptotic efficiency, concrete performance of implementations, and real-world applications. The focus of this paper is *recursive composition*, a notion that we motivate next.

Recursive composition. The time to validate a SNARG can be exponentially faster than the time to run the non-deterministic computation that it attests to, a property known as succinct verification. This exponential speedup raises an interesting prospect: could one produce a SNARG about a computation that involves validating prior SNARGs? Thanks to succinct verification, the time to run this (non-deterministic) computation would be essentially independent of the time of the prior computations. This *recursive composition* of SNARGs enables *incrementally verifiable computation* [Val08] and *proof-carrying data* [CT10; BCCT13]. A critical technicality here is that, for recursive composition to work, the SNARG must be an *argument of knowledge*, i.e., a SNARK. This is because the security of a SNARG holds only against efficient adversaries, and the knowledge property ensures that prior SNARGs must have been efficiently produced, and so we can rely in turn on their security. A formal treatment of this can be found in [BCCT13], which discusses how the "strength" of a SNARG's knowledge property relates to how many recursions the SNARG supports.

Efficient recursion. Theory tells us that *any* succinct-verifier SNARK is recursively composable [BCCT13]. In practice, however, recursive composition is exceedingly difficult to realize efficiently. The reason is that, even if we have a SNARK that is concretely efficient when used "standalone", it is often prohibitively expensive to express the SNARK verifier's computation through the language supported by the SNARK. Indeed, while by now there are numerous SNARK constructions with remarkable concrete efficiency, *to date there is only a single efficient approach to recursion*. The approach, due to [BCTV14], uses pairing-based SNARKs with a special algebraic property discussed below.[1] This has enabled real-world applications such as Coda [Co17], a cryptocurrency that uses recursive composition to achieve strong scalability properties.

Limitations. The above efficient approach to recursion suffers from significant limitations.

- *It is pre-quantum.* Pairing-based SNARKs rely (at least) on the hardness of extracting discrete logarithms, and so are insecure against quantum attacks. Hence the approach of [BCTV14] is also insecure against quantum attacks. Devising an efficient *post-quantum* approach to recursion is an open problem.
- *It introduces toxic waste.* All known pairing-based SNARKs that can be used in the approach of [BCTV14] rely on a structured reference string (SRS). Sampling the SRS involves secret values (the "toxic waste") that must remain secret for security. Ensuring that this is the case in practice is difficult: the SRS must be sampled by some trusted party or via a cryptographic ceremony [BCGTV15; BGG17; BGM17; ABLSZ19]. Devising an efficient *transparent* (toxic-waste free) approach to recursion is an open problem.
- *It uses expensive algebra.* The approach of [BCTV14] uses pairing-based SNARKs instantiated via *pairing-friendly cycles of elliptic curves*. Only a *single* cycle construction is known, *MNT cycles*; it consists of two prime-order elliptic curves, with embedding degrees 4 and 6 respectively. Curves in an MNT cycle must be much bigger than usual in order to compensate for the loss of security caused by the small embedding degrees. Moreover the fields that arise from MNT cycles are *imposed on applications* rather than being chosen depending on the needs of applications, causing additional performance overheads. Attempts to find "better" cycles, without these limitations, have resulted in some negative results [CCW19]. Indeed, finding *any other* cycles beyond MNT cycles is a challenging open problem.

[1] A recent note sketches an alternative approach to recursion based on batch verification [BGH19]. We omit a discussion of this note due to lack of sufficient detail (it does not provide definitions, full constructions, security arguments, or an efficiency analysis).

1.1 Our results

We present a new methodology for recursive composition that simultaneously overcomes all of the limitations discussed above. We experimentally validate our methodology, demonstrating feasibility in practice.

The starting point of our work is the observation that recursive composition is simpler when applied to a SNARG (of knowledge) that supports *preprocessing*, as we explain in Section 2.1. This property of a SNARG means that in an offline phase one can produce a short summary for a given circuit and then, in an online phase, one may use this short summary to verify SNARGs that attest to the satisfiability of the circuit with different partial assignments to its inputs. The online phase can be as fast as reading the SNARG (and the partial assignment), and in particular sublinear in the circuit size *even for arbitrary circuits*. Throughout, by "preprocessing SNARG" we mean a SNARG whose verifier runs in time *polylogarithmic* in the circuit size.[2]

Our methodology has three parts: (1) a transformation that maps any "holographic proof" into a preprocessing SNARG in the random oracle model; (2) a holographic proof for (rank-1) constraint systems, which leads to a corresponding preprocessing SNARG; (3) a transformation that recurses any preprocessing SNARK (once the random oracle is heuristically instantiated via a cryptographic hash function).

We now summarize our contributions for each of these parts.

(1) From holographic proofs to preprocessing SNARGs. A probabilistic proof is *holographic* if the verifier does not receive the circuit description as an input but, rather, makes a small number of queries to an encoding of the circuit [BFLS91]. Recent work [CHMMVW19] has established a connection between holography and preprocessing (which we review in Section 1.2). The theorem below adds to this connection, by showing that interactive oracle proofs (IOPs) [BCS16; RRR16] that are holographic can be compiled into preprocessing SNARGs that are secure in the quantum random oracle model [BDFLSZ11; CMS19].

Theorem 1 (informal)**.** *There is an efficient transformation that compiles any holographic IOP for a relation $\mathcal{R}$ into a preprocessing SNARG for $\mathcal{R}$ that is unconditionally secure in the random oracle model. If the IOP is a (honest-verifier) zero knowledge proof of knowledge then the transformation produces a zero knowledge SNARG of knowledge (zkSNARK). This extends to hold in the quantum random oracle model.*

By applying Theorem 1 to known holographic proofs for non-deterministic computations (such as the PCP in [BFLS91] or the IPCP in [GKR15]), we obtain the first transparent preprocessing SNARG and the first post-quantum preprocessing SNARG. Unfortunately, known holographic proofs are too expensive for practical use, because encoding the circuit is costly (as explained in Section 1.2.1). In this paper we address this problem by constructing an efficient holographic proof, discussed below.

We note that holographic proofs involve relations $\mathcal{R}$ that consist of *triples* rather than *pairs* because the statement being checked has two parts. One part is called the *index*, which is encoded in an offline phase by the *indexer* and this encoding is provided as an oracle to the verifier. The other part is called the *instance*, which is provided as an explicit input to the verifier. For example, the index may be a circuit description and the instance a partial assignment to its inputs. We refer to this notion as *indexed relations* (see Section 3.2).

(2) Efficient protocols for R1CS. We present a holographic IOP for rank-1 constraint satisfiability (R1CS), a standard generalization of arithmetic circuits where the "circuit description" is given by coefficient matrices. We describe the corresponding indexed relation.

Definition 1 (informal)**.** *The indexed relation $\mathcal{R}_{\mathrm{R1CS}}$ is the set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, n, m, A, B, C), x, w\big)$ where $\mathbb{F}$ is a finite field, A, B, C are $n \times n$ matrices over $\mathbb{F}$, each containing at most m non-zero entries, and $z := (x, w)$ is a vector in $\mathbb{F}^n$ such that $Az \circ Bz = Cz$. (Here "$\circ$" denotes the entry-wise product.)*

[2] In contrast, *non*-preprocessing SNARGs can achieve fast verification *only for structured circuits*, because the verification procedure must at a minimum read the *description* of the circuit whose satisfiability it checks. The description of a circuit can be much smaller than the circuit itself only when the circuit has suitable structure, e.g., repeated sub-components in parallel or in series.

Theorem 2 (informal)**.** *There exists a public-coin holographic IOP for the indexed relation $\mathcal{R}_{\mathrm{R1CS}}$ that is a zero knowledge proof of knowledge with the following efficiency features. In the offline phase, the encoding of an index is computable in $O(m \log m)$ field operations and consists of $O(m)$ field elements. In the online phase, the protocol has $O(\log m)$ rounds, with the prover using $O(m \log m)$ field operations and the verifier using $O(|x| + \log m)$ field operations. Proof length is $O(m)$ field elements and query complexity is $O(\log m)$.*

The above theorem improves, in the holographic setting, on prior IOPs for R1CS (see Fig. 1): it offers an exponential improvement in verification time compared to the linear-time verification of [BCRSVW19], and it offers succinct verification for all coefficient matrices compared to only structured ones as in [BCGGRS19].

Armed with an efficient holographic IOP, we use our compiler to construct an efficient preprocessing SNARG in the random oracle model. The following theorem is obtained by applying Theorem 1 to Theorem 2.

Theorem 3 (informal)**.** *There exists a preprocessing zkSNARK for R1CS that is unconditionally secure in the random oracle model (and the quantum random oracle model) with the following efficiency features. In the offline phase, anyone can publicly preprocess an index in time $O_\lambda(m \log m)$, obtaining a corresponding verification key of size $O_\lambda(1)$. In the online phase, the SNARG prover runs in time $O_\lambda(m \log m)$ and the SNARG verifier runs in time $O_\lambda(|x| + \log^2 m)$; argument size is $O_\lambda(\log^2 m)$.*

We have implemented the protocol underlying Theorem 3, obtaining the first efficient realization of a post-quantum transparent preprocessing zkSNARK.

For example, for a security level of 128 bits over a 181-bit prime field, arguments range from $80\,\mathrm{kB}$ to $200\,\mathrm{kB}$ for instances of up to millions of constraints. These argument sizes are two orders of magnitude bigger than *pre*-quantum *non*-transparent preprocessing zkSNARKs (see Section 1.2.2), and are 2× bigger that the state of the art in post-quantum transparent *non*-preprocessing zkSNARKs [BCRSVW19]. Our proving and verification times are comparable to prior work: proving takes several minutes, while verification takes several milliseconds *regardless of the constraint system*. (See Section 13 for performance details.)

Besides its application to post-quantum transparent recursion, our preprocessing zkSNARK provides attractive benefits over prior constructions, as we discuss in Section 1.2.2.

Note that, when the random oracle in the construction is heuristically instantiated via an efficient cryptographic hash function (as in our implementation), the resulting preprocessing zkSNARK is in the uniform reference string (URS) model, which means that the system parameters consist of a uniformly random string of fixed size.[3] The term "transparent" refers to a construction in the URS model.

(3) Post-quantum transparent recursion. We obtain the first efficient realization of post-quantum transparent recursive composition for SNARKs. The cryptographic primitive that formally captures this capability is known as *proof carrying data* (PCD) [CT10; BCCT13], and so this is what we construct.

Theorem 4 (informal)**.** *There is an efficient transformation that compiles any preprocessing SNARK in the URS model into a preprocessing PCD scheme in the URS model. Moreover, if the preprocessing SNARK is post-quantum secure then so is the preprocessing PCD scheme.*

The above transformation, which preserves the "transparent" property and post-quantum security, is where recursive composition occurs. For details, including the notion of PCD, see Section 11.

Moreover, we provide an efficient implementation of the transformation in Theorem 4 applied to our implementation of the preprocessing zkSNARK from Theorem 3. The main challenge is to express the SNARK verifier's computation in as few constraints as possible, and in particular to design a constraint

[3] We stress that this step is a heuristic due to well-known limitations to the random oracle methodology [CGH04; GK03]. Investigating how to provably instantiate the random oracle for many natural constructions is an active research frontier.

system for the SNARK verifier that on relatively small instances is smaller than the constraint system that it checks (thereby permitting arbitrary recursion depth). Via a combination of computer-assisted design and recent advances in algebraic hash functions, we achieve this threshold for all computations of at least 2 million constraints. Specifically, we can express a SNARK verifier checking 2 million constraints using only 1.7 million constraints, and this gap grows quickly with the computation size. *This is the first demonstration of post-quantum transparent recursive composition in practice.*

	R1CS		indexer	prover	verifier	round	proof	query
	instances	holographic?	time	time	time	complexity	length	complexity
[BCRSVW19]	arbitrary	NO	N/A	$O(m + n \log n)$	$O(\lvert\mathbb{x}\rvert + m)$	$O(\log n)$	$O(n)$	$O(\log n)$
[BCGGRS19] †	semi-succinct	NO	N/A	$O(m + n \log n)$	$O(\lvert\mathbb{x}\rvert + \log n)$	$O(\log n)$	$O(n)$	$O(\log n)$
this work	arbitrary	YES	$O(m \log m)$	$O(m \log m)$	$O(\lvert\mathbb{x}\rvert + \log m)$	$O(\log m)$	$O(m)$	$O(\log m)$

Figure 1: Comparison of IOPs for R1CS: two prior non-holographic IOPs, and our holographic IOP. Here n denotes the number of variables and m the number of non-zero coefficients in the matrices.
†: The parameters stated for [BCGGRS19] reflect replacing the constant-query low-degree test in the construction with a concretely-efficient logarithmic-query low-degree test such as [BBHR18], to simplify comparison.

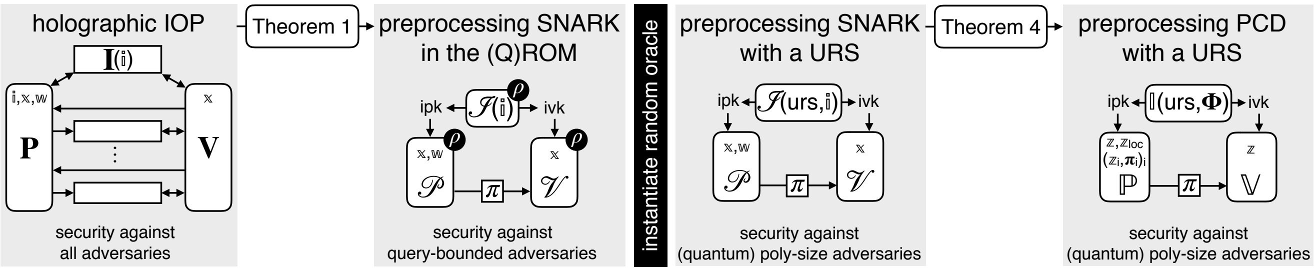

Figure 2: Diagram of our methodology to recursive composition that is post-quantum and transparent.

1.2 Comparison with prior work

We provide a comparison with prior work in the three areas to which we contribute: holographic proofs (Section 1.2.1); preprocessing SNARGs (Section 1.2.2); and recursive composition of SNARKs (Section 1.2.3). We omit a general discussion of the now ample literature on SNARGs, and in particular do not discuss *non*-preprocessing SNARGs for structured computations (e.g., [XZZPS19], [BBHR19], and many others).

1.2.1 Prior holographic proofs

The verifier in a proof system cannot run in time that is sublinear in its input, because it must at a minimum read the input in order to know the statement being checked. Holographic proofs [BFLS91] avoid this limitation by considering a setting where the verifier does not receive its input explicitly but, instead, has query access to an encoding of it. The goal is then to verify the statement in time *sublinear* in its size; note that such algorithms are necessarily probabilistic.[4]

[4] The goal of sublinear verification via holographic proofs is similar to, but distinct from, the goal of sublinear verification via *proximity proofs* (as, e.g., studied in [EKR04; DR04; BGHSV06; RVW13; GR15].) In this latter setting, the verifier has oracle access to an input that is *not* promised to be encoded and, in particular, cannot in general decide if the input is in the language without

In Fig. 3 we compare the efficiency of prior holographic proofs and our holographic proof for the case of circuit satisfiability, where the input to the verifier is the description of an arbitrary circuit. There are two main prior holographic proofs in the literature. One is the PCP construction in [BFLS91], where it suffices for the verifier to query a few locations of a low-degree extension of the circuit description. Another one is the "bare bones" protocol in [GKR15], which is a holographic IP for circuit *evaluation* that can be re-cast as a holographic IPCP for circuit *satisfaction*; the verifier relies on the low-degree extensions of functions that describe each layer of the circuit. The constructions in [BFLS91] and [GKR15] are unfit for practical use as holographic proofs in Theorem 1, because encoding the circuit incurs a polynomial blowup due to the use of *multivariate* low-degree extensions (which yield encodings with inverse polynomial rate).

In the table we exclude the "algebraic holographic proof" of Marlin [CHMMVW19], because the soundness guarantee of such a proof is incompatible with Theorem 1.

Comparison with this work. Our holographic proof is the first to achieve efficient asymptotics not only for the prover and verifier, but also for the indexer, which is responsible for producing the encoding of the circuit.

	proof	indexer	prover	verifier
	type	time	time	time
[BFLS91]	PCP	$\mathsf{poly}(N)$	$\mathsf{poly}(N)$	$\mathsf{poly}(\lvert\mathbb{x}\rvert + \log(N))$
[GKR15]	IPCP	$\mathsf{poly}(N)$	$\mathsf{poly}(\lvert\mathbb{w}\rvert) + O(N)$	$O(\lvert\mathbb{x}\rvert + D\log W)$
this work	IOP	$O(N\log N)$	$O(N\log N)$	$O(\lvert\mathbb{x}\rvert + \log N)$

Figure 3: Comparison of holographic proofs for arithmetic circuit satisfiability. Here $\mathbb{x}$ denotes the known inputs, $\mathbb{w}$ the unknown inputs, and N the total number of gates; if the circuit is layered, D denotes circuit depth and W circuit width. Our Theorem 1 can be used to compile any of these holographic proofs into a preprocessing SNARG. (For better comparison with other works, [GKR15] is stated as an IPCP for circuit satisfiability rather than as an IP for circuit evaluation; in the latter case, the prover time would be $O(N)$. The prover times for [GKR15] incorporate the techniques for linear-time sumcheck introduced in [XZZPS19].)

1.2.2 Prior preprocessing SNARGs

Prior works construct preprocessing SNARGs in a model where a trusted party samples, in a parameter setup phase, a structured reference string (SRS) that is proportional to circuit size. We summarize the main features of these constructions, distinguishing between the case of circuit-specific SRS and universal SRS.

- *Circuit-specific SRS:* a circuit is given as input to the setup algorithm, which samples a (long) proving key and a (short) verification key that can be used to produce and validate arguments for the circuit. Preprocessing SNARGs with circuit-specific SRS originate in [Gro10; Lip12; GGPR13; BCIOP13], and have been studied in an influential line of work that has led to highly-efficient constructions (e.g., [Gro16]) and large-scale deployments (e.g., [Zc14]). They are obtained by combining linear interactive proofs and linear-only encodings. The argument sizes achievable in this setting are very small: less than 200 bytes.

- *Universal SRS:* a size bound is given as input to the setup algorithm, which samples a (long) proving key and a (short) verification key that can be used to produce and validate arguments for circuits within this bound. A public procedure can then be used to specialize both keys for arguments relative to the desired circuit. Preprocessing SNARGs with universal (and updatable) SRS were introduced in [GKMMM18], and led

reading all of the input. To allow for sublinear verification without any promises on the input, the decision problem is relaxed: the verifier is only asked to decide if the input is in the language or *far* from any input in the language.

to efficient constructions in [MBKM19; CHMMVW19; GWC19]. They are obtained by combining "algebraic" holographic proofs (see below) and polynomial commitment schemes. The argument sizes currently achievable with universal SRS are bigger than with circuit-specific SRS: less than 1500 bytes.

Comparison with this work. Theorem 1 provides a methodology to obtain preprocessing SNARGs in the (quantum) random oracle model, which heuristically implies (by suitably instantiating the random oracle) preprocessing SNARGs that are post-quantum and transparent. Neither of these properties is achieved by prior preprocessing SNARGs. Theorem 1 also develops the connection between holography and preprocessing discovered in [CHMMVW19], which considers the case of holographic proofs where the completeness and soundness properties are restricted to "algebraic provers" (which output polynomials of prescribed degrees). We consider the case of general holographic proofs, where completeness and soundness are not restricted.

Moreover, our holographic proof (Theorem 2) leads to a preprocessing SNARG (Theorem 3) that, as supported by our implementation, provides attractive benefits over prior preprocessing SNARGs.

- Prior preprocessing SNARGs require cryptographic ceremonies to securely sample the long SRS, which makes deployments difficult and expensive. This has restricted the use of preprocessing SNARGs to proving relatively small computations, due to the prohibitive cost of securely sampling SRSs for large computations. This is unfortunate because preprocessing SNARGs could be useful for "scalability applications", which leverage succinct verification to efficiently check large computations (e.g., verifying the correctness of large batches of trades executed at a non-custodial exchange [RU19; SD19]).

  The transparent property of our preprocessing SNARG means that the long SRS is replaced with a fixed-size URS (uniform reference string). This simplifies deployments and enables scalability applications.

- Prior preprocessing SNARGs are limited to express computations over the prime fields that arise as the scalar fields of pairing-friendly elliptic curves. Such fields are imposed by parametrized curve families that offer little flexibility for optimizations or applications. (Alternatively one can use the Cocks–Pinch method [FST10] to construct an elliptic curve with a desired scalar field, but the resulting curve is inefficient.)

  In contrast, our preprocessing SNARG is easily configurable across a range of security levels, and supports most large prime fields and all large binary fields, which offers greater flexibility in terms of performance optimizations and customization for applications.

Remark 1.1 (weaker forms of preprocessing)**.** Prior work proved recursive composition only for non-interactive arguments of knowledge with succinct verifiers [BCCT13]; this is the case for our definition of preprocessing SNARGs. In this paper we show that recursive composition is possible even when the verifier is merely *sublinear* in the circuit size (see Section 11), though the cost of each recursion is much steeper than in the polylogarithmic case.

This provides additional motivation to the study of preprocessing with sublinear verifiers, as recently undertaken by Setty [Set19]. In this latter work, Setty proposes a non-interactive argument in the URS (uniform reference string) model where, for n-gate arithmetic circuits and a chosen constant $c \geq 2$, proving time is $O_\lambda(n)$, argument size is $O_\lambda(n^{1/c})$, and verification time is $O_\lambda(n^{1-1/c})$.

1.2.3 Recursion for pairing-based SNARKs

The approach to recursive composition of [BCTV14] uses pairing-based (preprocessing) SNARKs based on pairing-friendly cycles of elliptic curves. This approach applies to constructions with circuit-specific SRS (e.g. [Gro16]) *and* to those with universal SRS (e.g. [GKMMM18; MBKM19; CHMMVW19; GWC19]).

Informally, pairing-based SNARKs support languages that involve the satisfiability of constraint systems over a field that is *different* from the field used to compute the SNARK verifier — this restriction arises from the mathematics of the underlying pairing-friendly elliptic curve used to instantiate the pairing. This seemingly mundane fact has the regrettable consequence that expressing the SNARK verifier's computation in the language supported by the SNARK (to realize recursive composition) is unreasonably expensive due to this "field mismatch". To circumvent this barrier, prior work leveraged *two* pairing-based SNARKs where the field to compute one SNARK verifier equals the field of the language supported by the other SNARK, and vice versa. This condition enables each SNARK to efficiently verify the other SNARK's proofs.

These special SNARKs rely on pairing-friendly cycles of elliptic curves, which are pairs of pairing-friendly elliptic curves where the base field of one curve equals the scalar field of the other curve and vice versa. The only known construction is *MNT cycles*, which consist of two prime-order elliptic curves with embedding degrees 4 and 6 respectively. An MNT cycle must be much bigger than usual in order to compensate for the low security caused by the small embedding degrees. For example, for a security level of 128 bits, curves in an MNT cycle must be defined over a prime field with roughly 800 bits; this is over *three times* the 256 bits that suffice for curves with larger embedding degrees. These performance overheads can be significant in practice, e.g., Coda [Co17] is a project that has deployed MNT cycles in a product, and has organized a community challenge to speed up the proof generation for pairing-based SNARKs [SN]. A natural approach to mitigate this problem would be to find "high-security" cycles (i.e., with higher embedding degrees) but to date little is known about pairing-friendly cycles beyond a few negative results [CCW19].

Comparison with this work. The approach to recursion that we present in this paper is *not tied* to constructions of pairing-friendly cycles of elliptic curves. In particular, our approach scales gracefully across different security levels, and also offers more flexibility when choosing the desired field for an application. In addition, our approach is post-quantum and, moreover, uses a transparent (i.e., public-coin) setup.

On the other hand, our approach has two disadvantages. First, argument size is about 100 times bigger than the argument size achievable by cycle-based recursion. Second, the number of constraints needed to express the verifier's computation is about 45 times bigger than those needed in the case of cycle-based recursion (e.g., the verifier of [Gro16] can be expressed in about 40,000 constraints). The vast majority of these constraints come from the many hash function invocations required to verify the argument.

Both of the above limitations are somewhat orthogonal to our approach and arguably temporary: the large proof size and many hash invocations come from the many queries required from current constructions of low-degree tests [BBHR18; BGKS19]. As the state of the art in low-degree testing progresses (e.g., to high-soundness constructions over large alphabets), both argument size and verifier size will also improve.

2 Techniques

We discuss the main ideas behind our results. In Section 2.1 we explain how preprocessing simplifies recursive composition. In Section 2.2 we describe our compiler from holographic IOPs to preprocessing SNARGs (Theorem 1). In Section 2.3 we describe our efficient holographic IOP (Theorem 2), and then in Section 2.4 we discuss the corresponding preprocessing SNARG (Theorem 3). In Section 2.5 we describe how to obtain post-quantum and transparent PCD (Theorem 4). In Section 2.6 we discuss our verifier circuit.

Recall that indexed relations consist of *triples* $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ is the instance, and $\mathbb{w}$ is the witness (see Section 3.2). We use these relations because the statements being checked have two parts, the index $\mathbb{i}$ (e.g., a circuit description) given in an offline phase and the instance $\mathbb{x}$ (e.g., a partial input assignment) given in an online phase.

2.1 The role of preprocessing SNARKs in recursive composition

We explain why preprocessing simplifies recursive composition of SNARKs. For concreteness we consider the problem of incrementally proving the iterated application of a circuit $F \colon \{0,1\}^n \to \{0,1\}^n$ to an initial input $z_0 \in \{0,1\}^n$. We are thus interested in proving statements of the form "*given z_T there exists z_0 such that $z_T = F^T(z_0)$*", but wish to avoid having the SNARK prover check the correctness of all T invocations at once. Instead, we break the desired statement into T smaller statements $\{$"$z_i = F(z_{i-1})$"$\}_{i=1}^{T}$ and then inductively prove them. Informally, for $i = 1, \dots, T$, we produce a SNARK proof π_i for this statement:

> "*Given a counter i and claimed output z_i, there exists a prior output z_{i-1} such that $z_i = F(z_{i-1})$ and, if $i > 1$, there exists a SNARK proof π_{i-1} that attests to the correctness of z_{i-1}.*"

Formalizing this idea requires care, and in particular depends on how the SNARK achieves succinct verification (a prerequisite for recursive composition). There are two methods to achieve succinct verification.

(1) *Non-preprocessing SNARKs for structured computations.* The SNARK supports non-deterministic computations expressed as programs, i.e., it can be used to prove/verify statements of the form "given a program M, primary input $\mathbb{x}$, and time bound t, there exists an auxiliary input $\mathbb{w}$ such that M accepts $(\mathbb{x}, \mathbb{w})$ in t steps". (More generally, the SNARK could support any computation model for which the description of a computation can be significantly smaller than the size of the described computation.)

(2) *Preprocessing SNARKs for arbitrary computations.* The SNARK supports circuit satisfiability, i.e., it can be used to prove/verify statements of the form "given a circuit C and primary input $\mathbb{x}$, there exists an auxiliary input $\mathbb{w}$ such that $C(\mathbb{x}, \mathbb{w}) = 0$". Preprocessing enables the circuit C to be summarized into a short verification key ivk_C that can be used for succinct verification *regardless* of the structure of C. (More generally, the SNARK could support any computation model as long as preprocessing is possible.)

We compare the costs of recursive composition in these two cases, showing why the preprocessing case is cheaper. Throughout we consider SNARKs in the uniform reference string model, i.e., parameter setup consists of sampling a fully random string urs of size $\mathsf{poly}(\lambda)$ that suffices for proving/verifying any statement.

(1) Recursion without preprocessing. Let $(\mathcal{P}, \mathcal{V})$ be a *non*-preprocessing SNARK for non-deterministic program computations. In this case, recursion is realized via a program R, which depends on urs and F, that checks one invocation of the circuit F and the validity of a prior SNARK proof relative to the reference string urs. The program R is defined as follows:

Primary input: a tuple $\mathbb{x} = (M, i, z_i)$ consisting of the description of a program M, counter i, and claimed output z_i. (We later set $M := R$ to achieve recursion, as explained shortly.)

Auxiliary input: a tuple $\mathbb{w} = (z_{i-1}, \pi_{i-1})$ consisting of a previous output z_{i-1} and corresponding SNARK proof π_{i-1} that attests to its correctness.

Code: $R(\mathbb{x}, \mathbb{w})$ accepts if $z_i = F(z_{i-1})$ and, if $i > 1$, $\mathcal{V}(\mathsf{urs}, M, \mathbb{x}_{i-1}, t, \pi_{i-1}) = 1$ where $\mathbb{x}_{i-1} := (M, i-1, z_{i-1})$ and t is a suitably chosen time bound.

The program R can be used to incrementally prove the iterated application of the circuit F. Given a tuple $(i-1, z_{i-1}, \pi_{i-1})$ consisting of the current counter, output, and proof, one can use the SNARK prover to obtain the next tuple (i, z_i, π_i) by setting $z_i := F(z_{i-1})$ and computing the proof $\pi_i := \mathcal{P}(\mathsf{urs}, R, \mathbb{x}_i, t, \mathbb{w}_i)$ for the instance $\mathbb{x}_i := (R, i, z_i)$ and witness $\mathbb{w}_i := (z_{i-1}, \pi_{i-1})$ (and a certain time bound t). Note that we have set $M := R$, so that (the description of) R is part of the primary input to R. A tuple (i, z_i, π_i) can then be verified by running the SNARK verifier, as $\mathcal{V}(\mathsf{urs}, R, \mathbb{x}_i, t, \pi_i)$ for $\mathbb{x}_i := (R, i, z_i)$.[5]

We refer the reader to [BCCT13] for details on how to prove the above construction secure. The aspect that we are interested to raise here is that the program R is tasked to simulate itself, essentially working as a universal machine. This means that every elementary operation of R, and in particular of F, needs to be simulated by R in its execution. This essentially means that the computation time of R, which dictates the cost of each proof composition, is at least a constant $c > 1$ times the size of $|F|$. *This multiplicative overhead on the size of the circuit F, while asymptotically irrelevant, is a significant overhead in concrete efficiency.*

(2) Recursion with preprocessing. We describe how to leverage preprocessing in order to avoid universal simulation, and in particular to avoid *any* multiplicative performance overheads in recursive composition. Intuitively, preprocessing provides a "cryptographic simplification" to the requisite recursion, by enabling us to replace the description of the computation with a succinct cryptographic commitment to it.

Let $(\mathcal{I}, \mathcal{P}, \mathcal{V})$ be a preprocessing SNARK for circuits. Recursion is realized via a circuit R that depends on urs and F, and checks one invocation of F and a prior proof. The circuit R is defined as follows:

Primary input: a tuple $\mathbb{x} = (\mathsf{ivk}, i, z_i)$ consisting of an index verification key ivk, counter i, and claimed output z_i. (We later set $\mathsf{ivk} := \mathsf{ivk}_R$ to achieve recursion.)

Auxiliary input: a tuple $\mathbb{w} = (z_{i-1}, \pi_{i-1})$ consisting of a previous output z_{i-1} and corresponding SNARK proof π_{i-1} that attests to its correctness.

Code: $R(\mathbb{x}, \mathbb{w})$ accepts if $z_i = F(z_{i-1})$ and, if $i > 1$, $\mathcal{V}(\mathsf{urs}, \mathsf{ivk}, \mathbb{x}_{i-1}, \pi_{i-1}) = 1$ where $\mathbb{x}_{i-1} := (\mathsf{ivk}, i-1, z_{i-1})$.

The circuit R can be used for recursive composition as follows. In the offline phase, we run the indexer $\mathcal{I}$ on the circuit R, obtaining a long index proving key ipk_R and a short index verification key ivk_R that can be used to produce and validate SNARKs with respect to the circuit R. Subsequently, in the online phase, one can use the prover $\mathcal{P}$ to go from a tuple $(i-1, z_{i-1}, \pi_{i-1})$ to a new tuple (i, z_i, π_i) by letting $z_i := F(z_{i-1})$ and $\pi_i := \mathcal{P}(\mathsf{urs}, \mathsf{ipk}_R, \mathbb{x}_i, \mathbb{w}_i)$ for the instance $\mathbb{x}_i := (\mathsf{ivk}_R, i, z_i)$ and witness $\mathbb{w}_i := (z_{i-1}, \pi_{i-1})$. Note that we have set $\mathsf{ivk} := \mathsf{ivk}_R$, so that the verification key ivk_R is part of the primary input to the circuit R. A tuple (i, z_i, π_i) can then be verified by running the SNARK verifier, as $\mathcal{V}(\mathsf{urs}, \mathsf{ivk}_R, \mathbb{x}_i, \pi_i)$ for $\mathbb{x}_i := (\mathsf{ivk}_R, i, z_i)$.

Crucially, the circuit R does *not* perform any universal simulation involving the circuit F, and in particular does not incur multiplicative overheads. Indeed, $|R| = |F| + |\mathcal{V}| = |F| + o(|F|)$. This was enabled by preprocessing, which let us provide the index verification key ivk_R as input to the circuit R.

[5] The astute reader may notice that we could have applied the Recursion Theorem to the program R to obtain a new program R^* that has access to its own code, and thereby simplify primary inputs from triples $\mathbb{x} = (M, i, z_i)$ to pairs $\mathbb{x} = (i, z_i)$. This, however, adds unnecessary complexity. Indeed, here we can rely on the SNARK verifier to provide R with its own code as part of the primary input, obviating this extra step. (For reference, the Recursion Theorem states that for every program $A(x, y)$ there is a program $B(y)$ that computes $A(\langle B \rangle, y)$, where the angle brackets emphasize that the first argument is the description of the program B.)

In fact, preprocessing is *already* part of the efficient approach to recursive composition in [BCTV14]. There the preprocessing SNARK uses a structured, rather than uniform, reference string but the benefits of preprocessing are analogous (even when the reference string depends on the circuit or a bound on it).

In summary: preprocessing SNARKs play an important role in efficient recursive composition. Our first milestone is post-quantum and transparent preprocessing SNARKs, which we then use to achieve post-quantum and transparent recursive composition.

2.2 From holographic proofs to preprocessing with random oracles

We describe the main ideas behind Theorem 1, which provides a transformation that compiles any holographic IOP for an indexed relation $\mathcal{R}$ into a corresponding preprocessing SNARG for $\mathcal{R}$. See Section 10 for details.

Warmup: holographic PCPs. We first consider the case of PCPs, a special case of IOPs. Recall that the Micali transformation [Mic00] compiles a (non-holographic) PCP into a (non-preprocessing) SNARG. We modify this transformation to compile a *holographic* PCP into a *preprocessing* SNARG, by using the fact that the SNARG verifier output by the Micali transformation invokes the PCP verifier as a black box.

In more detail, the main feature of a holographic PCP is that the PCP verifier does not receive the index as an explicit input but, rather, makes a small number of queries to an encoding of the index given as an oracle. If we apply the Micali transformation to the holographic PCP, we obtain a SNARG verifier that must answer queries by the PCP verifier to the encoded index. If we simply provided the index as an input to the SNARG verifier, then we cannot achieve succinct verification and so would not obtain a preprocessing SNARG. Instead, we let the SNARG indexer compute the encoded index, compute a Merkle tree over it, and output the corresponding root as an *index verification key* for the SNARG verifier. We can then have the SNARG prover extend the SNARG proof with answers to queries to the encoded index, certified by authentication paths relative to the index verification key. In this way the SNARG verifier can use the answers in the SNARG proof to answer the queries to the encoded index by the underlying PCP verifier.

This straightforward modification to the Micali transformation works: one can prove that if the soundness error of the holographic PCP is ϵ then the soundness error of the preprocessing SNARG is $t\epsilon + O(t^2 \cdot 2^{-\lambda})$ against t-query adversaries in the random oracle model. (A similar expression holds for quantum adversaries.)

General case: holographic IOPs. While efficient constructions of holographic PCPs are not known, in this paper we show how to construct an efficient holographic IOP (see Section 2.3). Hence we are actually interested in compiling holographic IOPs. In this case our starting point is the BCS transformation [BCS16], which compiles a (non-holographic) IOP into a (non-prepreprocessing) SNARG. We adopt a similar strategy as above: we modify the BCS transformation to compile a *holographic* IOP into a *preprocessing* SNARG, using the fact that the SNARG verifier output by the BCS transformation invokes the IOP verifier as a black box. Indeed, the main feature of a holographic IOP is the fact that the IOP verifier makes a small number of queries to an encoding of the index given as an oracle. Therefore the SNARG indexer can output the Merkle root of the encoded index as an index verification key, which subsequently the SNARG verifier can use to authenticate answers about the encoded index claimed by the SNARG prover.

An important technical difference here is the fact that the soundness error of the resulting preprocessing SNARG is not related to the soundness error of the holographic IOP but, instead, to its *state-restoration soundness* (SRS) error, a stronger notion of soundness introduced in [BCS16]. Namely, we prove that if the SRS error of the holographic PCP is $\epsilon_{\mathrm{sr}}(t)$ then the soundness error of the preprocessing SNARG is $\epsilon_{\mathrm{sr}}(t) + O(t^2 \cdot 2^{-\lambda})$. This phenomenon is inherited from the (unmodified) BCS transformation.

PoK and ZK. If the holographic IOP is a proof of knowledge, our transformation yields a preprocessing SNARG of knowledge (SNARK). If the holographic IOP is honest-verifier zero knowledge, the preprocessing

SNARG is statistical zero knowledge. These features are inherited from the BCS transformation.

2.3 An efficient holographic proof for constraint systems

We describe the main ideas behind Theorem 2, which provides an efficient construction of a holographic IOP for rank-1 constraint satisfiability (R1CS). See Definition 1 for the indexed relation representing this problem.

Our starting point: Marlin. Our construction borrows ideas from the *algebraic holographic proof* (AHP) underlying Marlin, a pairing-based zkSNARK due to [CHMMVW19]. An AHP is similar to a holographic IOP, except that the indexer and the prover (both honest and malicious) send *low-degree univariate polynomials* rather than evaluations of functions. The verifier may evaluate these polynomials at any point in the field.

To understand how AHPs and holographic IOPs differ, it is instructive to consider how one might construct a holographic IOP from an AHP. A natural approach is to construct the indexer and prover for the hIOP as follows: run the indexer/prover of the AHP, and whenever the indexer/prover outputs a polynomial, evaluate it and send this evaluation as the oracle. There are several issues with this approach. First, hIOPs require a stronger soundness guarantee: soundness must hold against malicious provers that send *arbitrary* oracles. Second, evaluating the polynomial requires selecting a set $L \subseteq \mathbb{F}$ over which to evaluate it. In general, since the verifier in the AHP may query any point in $\mathbb{F}$, we would need to take $L := \mathbb{F}$, which is prohibitively expensive for the indexer and prover if $\mathbb{F}$ is much larger than the instance size (as it often is, for both soundness and application reasons). Third, assuming that one manages to decouple L and $\mathbb{F}$, the soundness error of one invocation of the AHP will (at best) decrease with $1/|L|$ instead of $1/\mathbb{F}$, which requires somehow reducing the soundness error of the AHP to, say, $1/2^{\lambda}$, and simply re-running in parallel the AHP for $\lambda - \log |L|$ would be expensive in all relevant parameters.

The first issue could be resolved by composing the resulting protocol with a low-degree test. This introduces technicalities because we cannot hope to check that the oracle is exactly low-degree (as required in an AHP) — we can only check that the oracle is *close* to low-degree. The best way to resolve the second issue depends on the AHP itself, and would likely involve out-of-domain sampling [BGKS19]. Finally, resolving the third issue may not be possible in general (in fact, we do not see how resolve it for the AHP in Marlin.)

These above issues show that, despite some similarities, there are markedly *different* design considerations on hIOPs versus AHPs. For this reason, while we will follow some of the ideas outlined above, we do not take the Marlin AHP as a black box. Instead, we will draw on the ideas underlying the Marlin AHP in order to build a suitable hIOP for this paper. Along the way, we also show how to reduce the round complexity of the Marlin AHP from 3 to 2, an ideas that we use to significantly improve the efficiency of our construction.

Aurora. The structure of our holographic IOP, like the Marlin AHP, follows the one of Aurora [BCRSVW19], an IOP for R1CS that we now briefly recall. Given an R1CS instance (A, B, C), the prover sends to the verifier f_z, the RS-encoding of a vector z, and three oracles f_A, f_B, f_C which are purportedly the RS-encodings of the three vectors Az, Bz, Cz respectively. The prover and verifier then engage in subprotocols to prove that (i) f_A, f_B, f_C are indeed encodings of Az, Bz, Cz, and (ii) $f_A \cdot f_B - f_C$ is an encoding of the zero vector.

Together these checks ensure that (A, B, C) is a satisfiable instance of R1CS. Testing (ii) is a straightforward application of known probabilistic checking techniques, and can be achieved with a logarithmic-time verifier. The primary challenge in the Aurora protocol (and protocols based on it) is testing (i).

In the Aurora protocol this is achieved via a reduction to univariate sumcheck, a univariate analogue of the [LFKN92] sumcheck protocol. Univariate sumcheck also has a logarithmic verifier, but the reduction itself runs in time linear in the number of nonzero entries in the matrices A, B, C. A key technical contribution of the Marlin AHP is showing how to shift most of the cost of the reduction to the indexer in order to reduce the online cost of verification to logarithmic, as we now explain.

Challenges. We describe the original lincheck protocol of [BCRSVW19], and explain why it is not holographic. The lincheck protocol, on input a matrix $M \in \mathbb{F}^{k \times k}$ and RS-encodings of vectors $\vec{x}, \vec{y} \in \mathbb{F}^k$, checks whether $\vec{x} = M\vec{y}$. It makes use of the following two facts: (i) for a vector of linearly-independent polynomials $\vec{u} \in \mathbb{F}[X]^k$ and any vectors $\vec{x}, \vec{y} \in \mathbb{F}^k$, if $\vec{x} \neq \vec{y}$ then the polynomials $\langle \vec{u}, \vec{x} \rangle$ and $\langle \vec{u}, \vec{y} \rangle$ are distinct, and so differ with high probability at a random $\alpha \in \mathbb{F}$, and (ii) for any matrix $M \in \mathbb{F}^{k \times k}$, $\langle \vec{u}, M\vec{y} \rangle = \langle \vec{u}M, \vec{y} \rangle$. The lincheck verifier sends a random $\alpha \in \mathbb{F}$ to the prover, and the prover then convinces the verifier that $\langle \vec{u}M, \vec{y} \rangle(\alpha) - \langle \vec{u}, \vec{x} \rangle(\alpha) = 0$ using the univariate sumcheck protocol.

This requires the verifier to evaluate the low-degree extensions of $\vec{u}_\alpha$ and $\vec{u}_\alpha M$ at a point $\beta \in \mathbb{F}$, where $\vec{u}_\alpha \in \mathbb{F}^k$ is obtained by evaluating each entry of $\vec{u}$ at α. This is equivalent to evaluating the bivariate polynomials $u(X,Y), u_M(X,Y) \in \mathbb{F}[X,Y]$, obtained respectively by extending $\vec{u}, \vec{u}M$ over Y, at a random point in $(\alpha, \beta) \in \mathbb{F}^2$. By choosing $\vec{u}$ appropriately, we can ensure that $u(X,Y)$ can be evaluated in logarithmic time [BCGGRS19]. But, without help from an indexer, evaluating $u_M(\alpha, \beta)$ requires time $\Omega(\|M\|)$.

A natural suggestion in the holographic setting is to have the indexer evaluate u_M over some domain $S \subseteq \mathbb{F} \times \mathbb{F}$, and make this evaluation part of the encoded index. This does achieve the goal of logarithmic verification time. Unfortunately, the degree of u_M in each variable is about k, and so even writing down the coefficients of u_M requires time $\Omega(k^2)$, which for sparse M is quadratic in $\|M\|$.

In the Marlin lincheck the indexer instead computes a certain *linear-size* (polynomial) encoding of M, which the verifier then uses in a multi-round protocol with the prover to evaluate u_M at its chosen point. Our holographic lincheck improves upon this protocol, reducing the number of rounds by one; we describe it next.

Our holographic lincheck. Recall from above that the lincheck verifier needs to check that $\langle \vec{u}, \vec{x} \rangle$ and $\langle \vec{u}M, \vec{y} \rangle$ are equal as polynomials in X. To do this, it will choose a random $\alpha \in \mathbb{F}$ and send it to the prover, then engage in the univariate sumcheck protocol to show that $\sum_h u(\alpha, h)\hat{x}(h) - u_M(\alpha, h)\hat{y}(h) = 0$, where $\hat{x}, \hat{y}$ are low-degree extensions of x and y.

To verify the above sum, the verifier must compute $u(\alpha, \beta)$ and $u_M(\alpha, \beta)$ for some $\beta \in \mathbb{F}$. The former can be computed in by the verifier in logarithmic time as discussed; for the latter, we ask the prover to help. Specifically, we show that $u_M \equiv \hat{M}^*$, the unique bivariate low-degree extension of a matrix M^* which can be computed in quasilinear time from M (and in particular has $\|M^*\| = \|M\|$). Hence to show that $u_M(\alpha, \beta) = \gamma$ the prover and verifier can engage in a holographic *matrix arithmetization* protocol for M^* to show that $\hat{M}^*(\alpha, \beta) = \gamma$. Marlin makes use of a similar matrix arithmetization protocol, but for M itself, with a subprotocol to compute u_M from $\hat{M}$, which is a cost that we completely eliminate. Another improvement is that for our matrix arithmetization protocol we can efficiently reduce soundness error even when using a low-degree test, due to its non-recursive use of the sumcheck protocol.

Matrix arithmetization. Our matrix arithmetization protocol is a holographic IOP for computing the low-degree extension of a matrix $M \in \mathbb{F}^{H \times H}$ (provided in the index). It is useful here to view M in its sparse representation as a map $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$ for some $K \subseteq \mathbb{F}$, where if $\langle M \rangle(k) = (a, b, \gamma)$ for some $k \in K$ then $M_{a,b} = \gamma$, and $M_{a,b} = 0$ otherwise.

The indexer computes $\hat{\mathsf{row}}, \hat{\mathsf{col}}, \hat{\mathsf{val}}$ which are the unique low-degree extensions of the functions $K \to \mathbb{F}$ induced by restricting $\langle M \rangle$ to its first, second, and third coordinates respectively, and outputs their evaluations over L. It is not hard to verify that

$$\hat{M}(\alpha, \beta) = \sum_{k \in K} L_{H, \hat{\mathsf{row}}(k)}(\alpha) L_{H, \hat{\mathsf{col}}(k)}(\beta) \hat{\mathsf{val}}(k) \quad ,$$

for any $\alpha, \beta \in \mathbb{F}$, where $L_{H,a}$ is the polynomial of minimal degree which is 1 on a and 0 on $H \setminus \{a\}$. In order to check this equation using the sumcheck protocol we must modify the right-hand side: the summand must be a polynomial which can be efficiently evaluated. To this end, we make use of the “unnormalized

Lagrange" polynomial $u_H(X, Y) := (v_H(X) - v_H(Y))/(X - Y)$ from [BCGGRS19]. This polynomial has the property that for every $a, b \in H$, $u_H(a, b)$ is 0 if $a \neq b$ and nonzero if $a = b$; and it is easy to evaluate at every point in $\mathbb{F}$. By having the indexer renormalize $\hat{\mathsf{val}}$ appropriately, we obtain

$$\hat{M}(X, Y) \equiv \sum_{k \in K} u_H(\hat{\mathsf{row}}(k), \alpha) u_H(\hat{\mathsf{col}}(k), \beta) \hat{\mathsf{val}}(k) \ .$$

We have made progress, but now the summand has quadratic degree: $\Omega(|H||K|)$ because we *compose* the polynomials u_H and $\hat{\mathsf{row}}, \hat{\mathsf{col}}$. Next we show how to remove this composition.

Observe that since the image of K under $\hat{\mathsf{row}}, \hat{\mathsf{col}}$ is contained in H, $v_H(\hat{\mathsf{row}}(k)) = v_H(\hat{\mathsf{col}}(k)) = 0$. Hence the rational function

$$\frac{v_H(\alpha)}{(\alpha - \hat{\mathsf{row}}_{\langle M \rangle}(X))} \cdot \frac{v_H(\beta)}{(\beta - \hat{\mathsf{col}}_{\langle M \rangle}(X))} \cdot \hat{\mathsf{val}}_{\langle M \rangle}(X)$$

agrees with the summand on K; it is a rational extension of the summands. Moreover, the degrees of the numerator and denominator of the function are both $O(|K|)$. Now it remains to design a protocol to check the sum of a univariate rational function.

Rational sumcheck. Suppose that we want to check that $\sum_{k \in K} p(k)/q(k) = \gamma$, where p, q are low-degree polynomials. First, we have the prover send the (evaluation of the) unique polynomial f of degree $|K| - 1$ which agrees with p/q on K; that is, the unique low-degree extension of p/q viewed as a function from K to $\mathbb{F}$. We can use the *standard* univariate sumcheck protocol from [BCRSVW19] to test that $\sum_{k \in K} f(k) = \gamma$.

It then remains to check that f does indeed agree with p/q on K. This is achieved using standard techniques: if $p(k)/q(k) = f(k)$ for all $k \in K$, then $p(k) = q(k) \cdot f(k)$ for all $k \in K$ (at least if q does not vanish on K). Then $p - q \cdot f$ is a polynomial vanishing on K, and so is divisible by v_K. This can be checked using low-degree testing; for more details, see Section 5. Moreover, the degree of this equation is $\max(\deg(p), \deg(q) + |K|)$; in the matrix arithmetization protocol, this is $O(|K|)$.

Proof of knowledge and zero knowledge. Our full protocol for R1CS is a proof of knowledge, because when the verifier accepts with high enough probability it is possible to decode f_z into a satisfying assignment. We further achieve zero knowledge via techniques inherited from [BCRSVW19]. (Note that zero knowledge is not relevant for the matrix arithmetization protocol because the constraint matrices A, B, C are public.)

2.4 Post-quantum and transparent preprocessing

If we apply the compiler described in Section 2.2 (as captured in Theorem 1) to the efficient holographic proof for R1CS described in Section 2.3 (as captured in Theorem 2) then we obtain an efficient preprocessing zkSNARK for R1CS that is unconditionally secure in the (quantum) random oracle model (as captured in Theorem 3). We refer to the resulting construction as FRACTAL.

Implementation. We have implemented FRACTAL by extending the `libiop` library to support generic compilation of holographic proofs into preprocessing SNARGs, and then writing in code our holographic proof for R1CS. Our implementation supports a range of security levels and fields. (The only requirement on the field is that it contains certain smooth subgroups.) See Section 12.1 for more details on the implementation.

Clearly, the security of our implementation relies on the random oracle methodology applied to preprocessing SNARGs produced by our compiler, namely, we assume that if we replace every call to the random oracle with a call to a cryptographic hash function then the resulting construction, which formally is in the URS model, inherits the relevant security properties that we proved in the (quantum) random oracle model.

Evaluation. We have evaluated FRACTAL, and its measured performance is consistent with asymptotic predictions. In particular, the polylogarithmic argument size and verification time quickly become smaller than native witness size and native execution time as the size of the checked computation increases.

We additionally compare the costs of FRACTAL to prior preprocessing SNARGs, finding that (a) our prover and verifier times are comparable to prior constructions; (b) argument sizes are larger than prior constructions (that have an SRS). The larger argument sizes of FRACTAL are nonetheless comparable with other post-quantum transparent *non*-preprocessing SNARGs. See Section 13.1 for more details on evaluation.

2.5 Post-quantum and transparent recursive composition

We summarize the ideas behind our contributions to recursive composition of SNARKs.

Proof-carrying data. Recursive composition is captured by a cryptographic primitive called *proof-carrying data* (PCD) [CT10; BCCT13], which will be our goal. Consider a network of nodes, where each node receives messages from other nodes, performs some local computation, and sends the result on. PCD is a primitive that allows us to check the correctness of such distributed computations by recursively producing proofs of correctness for each message. Here "correctness" is locally specified by a *compliance predicate* Φ, which takes as input the messages received by a node and the message sent by that node (and possibly some auxiliary local data). A distributed computation is then considered Φ-*compliant* if, for each node, the predicate Φ accepts the node's messages (and auxiliary local data).

PCD captures proving the iterated application of a circuit as in Section 2.1, in which case the distributed computation evolves along a path. PCD also captures more complex topologies, which is useful for supporting distributed computations on long paths (via "depth-reduction" techniques [Val08; BCCT13]) and for expressing dynamic distributed computations (such as MapReduce computations [CTV15]).

From random oracle model to the URS model. While we have so far discussed constructions that are unconditionally secure in the (quantum) random oracle model, for recursion we now leave this model (by heuristically instantiating the random oracle with a cryptographic hash function) and start from preprocessing SNARKs in the URS model. The reason for this is far from mundane (and not motivated by implementation), as we now explain. The verifiers from Theorem 1 make calls to the random oracle, and therefore proving that the verifier has accepted would require using a SNARK that can prove the correctness of computations *in a relativized world where the oracle is a random function*. There is substantial evidence from complexity theory that such SNARKs do not exist (e.g., the PCP Theorem does not relativize with respect to a random oracle [CCRR92; For94]). By instantiating the random oracle, all oracle calls can be "unrolled" into computations that do not involve oracle gates, and thus we can prove the the correctness of the resulting computation.[6] We stress that random oracles cannot be securely instantiated in the general case [CGH04], and so we will assume that there is a secure instantiation of the random oracle for the preprocessing SNARKs produced via Theorem 1 (which, in particular, preserves proof of knowledge).

From SNARK to PCD. We prove that any preprocessing SNARK in the URS model can be transformed into a preprocessing PCD scheme in the URS model (Theorem 11.5).[7] The construction, described in Section 11, realizes recursive composition by following the template given in Section 2.1, except that the compliance predicate Φ may expect multiple input messages. This construction simplifies that of [BCCT13]

[6] The necessity to instantiate the random oracle before recursion also arises in the first construction of incrementally verifiable computation [Val08]. One way to circumvent this difficulty is to consider oracles that are equipped with a public verification procedure [CT10], however this requires embedding a secret in the oracle, which does not lend itself to straightforward software realizations and so we do not consider this approach in this paper.

[7] Analogously to a SNARK, here *preprocessing* denotes the fact that the PCD scheme enables succinct verification regardless of the computation expressed by the compliance predicate Φ (as opposed to only for structured computations).

for preprocessing SNARKs in the SRS model: we do not need to rely on collision-resistant hash functions to shrink the verification key ivk because we require it to be succinct, as captured in Lemma 11.8.[8]

Security against quantum adversaries. A key feature of our result (Theorem 11.5) is that we prove that if the SNARK is secure (i.e., is a proof of knowledge) against quantum adversaries then so is the resulting PCD scheme (i.e., it is also a proof of knowledge). Therefore, if we assume that FRACTAL achieves proof of knowledge against quantum adversaries when the random oracle is suitably instantiated, then by applying our result to FRACTAL we obtain a *post-quantum* preprocessing PCD scheme in the URS model.

We highlight here an important subtlety that arises when proving security against quantum adversaries. The proof of [BCCT13] makes use of the fact that, in the classical case, we may assume that the adversary is deterministic by selecting its randomness. This is not the case for quantum adversaries, since a quantum circuit can create its own randomness (e.g. by measuring a qubit in superposition). This means that we must be careful in defining the proof-of-knowledge property we require of the underlying SNARK. In particular, we must ensure that when we recursively extract proofs, these proofs are consistent with previously extracted proofs. When the adversary is deterministic, this is trivially implied by standard proof of knowledge; for quantum adversaries, it is not. We give a natural definition of proof of knowledge that suffices for the security reduction, and prove that it is realized by our SNARK construction (in the random oracle model).

2.6 The verifier as a constraint system

In order to recursively compose FRACTAL (the preprocessing zkSNARK discussed in Section 2.4), we need to express FRACTAL's verifier as a constraint system. The size of this constraint system is crucial because this determines the threshold at which recursive composition becomes possible. Towards this goal, we design and implement a constraint system that applies to a general class of verifiers, as outlined below. FRACTAL's verifier is obtained as an instantiation within this class. See Section 12.2 for details.

Hash computations introduced by the compiler. Our compiler (Theorem 1) transforms any holographic IOP into a corresponding preprocessing SNARG, while preserving relevant zero knowledge or proof of knowledge properties. The preprocessing SNARG verifier makes a black-box use of the holographic IOP verifier, which means that we can design a *single* (parametrized) constraint system representing the transformation that works for *any* holographic IOP. All additional computations introduced by the compiler involve cryptographic hash functions (which heuristically instantiate the random oracle). In particular, there are two types of hash computations: (1) a hash chain computation used to derive the randomness for each round of the holographic IOP verifier, based on the Merkle roots provided by the preprocessing SNARG prover; and (2) verification of Merkle tree authentication paths in order to ensure the validity of the query answers provided by the preprocessing SNARG prover. We design generic constraint systems for both of these tasks. Since we are designing constraint systems it is more efficient to consider multiple hash functions specialized to work in different roles: a hash function to absorb inputs or squeeze outputs in the hash chain; a hash function to hash leaves of the Merkle tree; a many-to-one hash function for the internal nodes of the Merkle tree; and others.

Choice of hash function. While our implementation is generic with respect to the aforementioned hash functions (replacing any one of them with another would be a rather straightforward task), the choice of hash function is nonetheless critical for concrete efficiency as we now explain. Expressing standard cryptographic hash functions, such as from the SHA or Blake family, as a constraint system requires more than 20,000 constraints. While this is acceptable for certain applications, these costs are prohibitive for hash-intensive

[8] In contrast, the verification key ivk in [BCCT13] is allowed to grow linearly with the public input to the circuit that it summarizes, and so recursion required replacing ivk with a short hash of it, and moving ivk to the witness of the recursion circuit.

computations, as is the case for the verifiers output by our compiler. Fortunately, the last few years have seen exciting progress in the design of *algebraic hash functions* [AD18; Alb+19a; GKKRRS19; AABSDS19; Alb+19b], which by design can be expressed via a small number of arithmetic constraints over large finite fields. While this is an active research front, and in particular no standards have been agreed upon, many of the proposed functions are *significantly cheaper* than prior ones, and their security analyses are promising. In this work we decide to use one of these as our choice of hash function (Rescue [AABSDS19]). We do not claim that this is the "best" choice among the currently proposed ones. (In fact, we know how to achieve better results via a combination of different choices.) We merely make one choice that we believe to be reasonable, and in particular suffices to demonstrate the feasibility of our methodology in practice.

Holographic IOP computations. The constraint system that represents the holographic IOP verifier will, naturally, depend on the specific protocol that is provided as input to the compiler.

That said, all known efficient IOPs, holographic or otherwise, are obtained as the combination of two ingredients: (1) a low-degree test for the Reed–Solomon (RS) code; and (2) an RS-encoded IOP, which is a protocol where the verifier outputs a set of algebraic claims, known as rational constraints, about the prover's messages. Examples of IOPs that fall in this category include our holographic IOP for R1CS, as well as protocols for R1CS in [AHIV17; BCRSVW19; BCGGRS19] and for AIRs in [BBHR19].

We thus provide two constraint systems that target these two components. First, we provide a constraint system that realizes the FRI low-degree test [BBHR18], which is used in many efficient IOPs, including in our holographic IOP for R1CS. Second, we provide infrastructure to write constraint systems that express a desired RS-encoded IOP. This essentially entails specifying how many random elements the verifier should send in each round of the protocol, and then specifying constraints that express the rational constraints output by the verifier at the end of the RS-encoded IOP.

We then use the foregoing infrastructure to express the verifier of our holographic IOP for R1CS as a constraint system. We note that the very same generic infrastructure would make it straightforward to express the verifiers of other protocols with the same structure [AHIV17; BBHR19; BCRSVW19; BCGGRS19].

Remark 2.1 (succinct languages)**.** Our work in writing constraints for the verifier is restricted to non-uniform computation models such as R1CS (i.e., we are not concerned about the global structure of the constraint system). We do not claim to have an efficient way to express the same verifier via succinct languages such as AIR [BBHR19] or Succinct-R1CS [BCGGRS19]. Doing so remains an open problem that, if addressed, may lead to additional opportunities in recursive composition (through *non*-preprocessing SNARKs).

3 Preliminaries

We state time costs in terms of basic operations over a given field $\mathbb{F}$, and size costs in terms of field elements in $\mathbb{F}$. We use the "big-oh" notation $O_{\mathbb{F}}$ to remind the reader that $\mathbb{F}$-operations and $\mathbb{F}$-elements have unit cost.

3.1 Sparse representations of matrices

Our protocols leverage sparse representations of matrices for efficiency, following the definition below. The definition is primarily for convenience in the sense that any reasonable sparse representation of a matrix can be transformed, in linear time, into one that follows the definition that we use.

Definition 3.1. *Let $H, K \subseteq \mathbb{F}$. A* **sparse representation** *of a matrix is a function $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$ that is injective when its output is restricted to $H \times H$. The matrix $M \in \mathbb{F}^{H \times H}$ is obtained from $\langle M \rangle$ by setting, for $a, b \in H$, $M_{a,b} := \gamma$ if there exists $k \in K$ such that $\langle M \rangle(k) = (a, b, \gamma)$ and $M_{a,b} := 0$ otherwise.*

Note that a matrix $M \in \mathbb{F}^{H \times H}$ has many possible sparse representations. In particular, we may choose any large enough K and any injection from K to $H \times H$ that "covers" the nonzero entries of M.

3.2 Indexed relations

An *indexed relation* $\mathcal{R}$ 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* $\mathcal{L}(\mathcal{R})$ is the set of pairs $(\mathbb{i}, \mathbb{x})$ for which there exists a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$. For example, the indexed relation of satisfiable boolean circuits consists of triples where $\mathbb{i}$ is the description of a boolean circuit, $\mathbb{x}$ is an assignment some of the input wires, and $\mathbb{w}$ is an assignment to the remaining input wires that makes the circuit output 0.

In this paper we build protocols for the indexed relation that represents *rank-1 constraint satisfiability* (R1CS), a generalization of arithmetic circuits where the "circuit description" is given by coefficient matrices.

Definition 3.2 (R1CS indexed relation)**.** *The indexed relation $\mathcal{R}_{\mathrm{R1CS}}$ is the set of all triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, H, K, \langle A \rangle, \langle B \rangle, \langle C \rangle), (I, x), w\big)$$

where $\mathbb{F}$ is a finite field, H, K are subsets of $\mathbb{F}$, $\langle A \rangle, \langle B \rangle, \langle C \rangle \colon K \to H \times H \times \mathbb{F}$ are sparse representations of $H \times H$ matrices over $\mathbb{F}$, I is a subset of H, $x \in \mathbb{F}^{I}$, $w \in \mathbb{F}^{H \setminus I}$, and $z := (x, w) \in \mathbb{F}^{H}$ is a vector such that $Az \circ Bz = Cz$. (Here "$\circ$" denotes the entry-wise product between two vectors.)

Remark 3.3. The above definition can be generalized to the case where the matrices are *non-square*, namely, the matrices are in $\mathbb{F}^{H_1 \times H_2}$ for possibly distinct domains $H_1, H_2 \subseteq \mathbb{F}$ [BCRSVW19]. All results stated in this paper extend to this non-square case. Our focus on the square case is only for simplicity of exposition.

3.3 Algebra

Polynomial encodings. For a finite field $\mathbb{F}$, subset $S \subseteq \mathbb{F}$, and function $f \colon S \to \mathbb{F}$ we denote by $\hat{f}$ the (unique) univariate polynomial over $\mathbb{F}$ with degree less than $|S|$ such that $\hat{f}(a) = f(a)$ for every $a \in S$. More explicitly, $\hat{f}(X) := \sum_{a \in S} f(a) L_{a,S}(X)$, where $L_{a,S}$ (for $a \in S$) is the unique (Lagrange) polynomial of degree less than $|S|$ such that $L_{a,S}(a) = 1$ and $L_{a,S}(b) = 0$ for all $b \in S \setminus \{a\}$.

Reed–Solomon code. Given a subset L of a field $\mathbb{F}$ and degree bound $d < |L|$, we denote by $\mathrm{RS}[L,d] \subseteq \mathbb{F}^L$ all evaluations over L of univariate polynomials of degree at most d:

$$\mathrm{RS}[L,d] := \left\{ f \colon L \to \mathbb{F} \text{ s.t. } \exists\, \hat{f} \in \mathbb{F}[X] \text{ with } \deg(\hat{f}) \leq d \text{ and } \hat{f}(L) = f \right\} \quad .$$

Whenever a polynomial $\hat{f}$ as above exists, then $\hat{f}$ is unique. The *rate* of $\mathrm{RS}[L,d]$ is $\rho := (d+1)/|L| \in (0,1)$, and its *distance* is $1-\rho$. The message encoded by $f \in \mathrm{RS}[L,d]$ is the restriction of $\hat{f}$ to a distinguished subset $H \subseteq \mathbb{F}$ of size $d+1$. (Note that H need not be a subset of L.) Observe that for every polynomial $\hat{f} \in \mathbb{F}[X]$ with degree less than $|L|$ it holds that the word $f_L := \hat{f}|_L$ is in $\mathrm{RS}[L, \deg(\hat{f})]$ (we will drop the subscript when the choice of domain L is clear from context). This means that there is a bijection between words in $\mathrm{RS}[L,d]$ and polynomials in $\mathbb{F}[X]$ of degree at most d.

We frequently move between univariate polynomials over $\mathbb{F}$ and their evaluations on domains $L \subseteq \mathbb{F}$. We use plain letters like f, g, h to denote functions from L to $\mathbb{F}$, and "hatted letters" $\hat{f}, \hat{g}, \hat{h}$ to denote the polynomials of minimal degree that agree with the corresponding functions on L. Conversely, if we "drop the hat" from a polynomial, then we consider its evaluation over L (which will always be larger than the degree).

Domains with subgroup structure. For a finite field $\mathbb{F}$, by "subgroup of $\mathbb{F}$" we mean either a subgroup of the additive group of $\mathbb{F}$ or a subgroup of $\mathbb{F}^*$. By "coset of $\mathbb{F}$" we mean a coset of a subgroup of $\mathbb{F}$ (additive or multiplicative). Throughout the paper we assume that the domain L for the Reed–Solomon code has "smooth" subgroup structure, meaning that it factors as a direct product of small (i.e., constant-size) subgroups. Under this assumption we can encode a message using the Reed–Solomon code in time $O_{\mathbb{F}}(|L| \log |L|)$. This assumption is also required by the low-degree test that we use [BBHR18; BGKS19].

Vanishing polynomials. For a finite field $\mathbb{F}$ and subset $S \subseteq \mathbb{F}$, we denote by v_S the unique non-zero monic polynomial of degree at most $|S|$ that is zero on S; v_S is called the *vanishing polynomial* of S. If S is a coset in $\mathbb{F}$ then the coefficients of v_S can be found in time $O_{\mathbb{F}}(\log^2 |S|)$, and subsequently v_S can be evaluated at any point in time $O_{\mathbb{F}}(\log |S|)$.[9] In the holographic setting we can have the indexer find v_S for any S of interest, so that the verifier can evaluate v_S in time $O_{\mathbb{F}}(\log |S|)$. In this paper we assume that this is the case, so that for any coset S in $\mathbb{F}$ we can evaluate its vanishing polynomial at any point in time $O_{\mathbb{F}}(\log |S|)$.

Derivative of vanishing polynomials. We rely on various properties of the bivariate polynomial u_S related to the formal derivative of v_S, first exploited to obtain efficient probabilistic proofs in [BCGGRS19]. For a finite field $\mathbb{F}$ and subset $S \subseteq \mathbb{F}$, we define

$$u_S(X,Y) := \frac{v_S(X) - v_S(Y)}{X - Y} \quad ,$$

which is a polynomial of individual degree $|S|-1$ because $X-Y$ divides $X^i - Y^i$ for any positive integer i. Note that $u_S(X,X)$ is the formal derivative of the vanishing polynomial $v_S(X)$.[10]

The bivariate polynomial $u_S(X,Y)$ satisfies two useful algebraic properties. First, it is strongly related to the Lagrange polynomials $L_{a,S}$ for $a \in S$. Specifically, $u_S(X,a) \equiv u_S(a,X) \equiv L_{a,S}(X) \cdot u_S(a,a)$ for all $a \in S$. In particular, this implies that the polynomials $(u_S(X,a))_{a \in S}$ are linearly independent. Second, the (unique) low-degree extension (in Y) of the vector $(u_S(X,a))_{a \in S} \in \mathbb{F}[X]^S$ is precisely $u_S(X,Y)$.

[9] If S is a multiplicative subgroup of $\mathbb{F}$ then $v_S(X) = X^{|S|} - 1$. More generally, if S is a ξ-coset of a multiplicative subgroup S_0 (namely, $S = \xi S_0$) then $v_S(X) = \xi^{|S|} v_{S_0}(X/\xi) = X^{|S|} - \xi^{|S|}$. In either case, v_S can be found and then evaluated at any point in time $O_{\mathbb{F}}(\log |S|)$. If instead S is an additive subgroup then there is an algorithm to find the coefficients of v_S in time $O_{\mathbb{F}}(\log^2 |S|)$. Being a linearized polynomial, v_S has only $O(\log |S|)$ nonzero coefficients, and in particular can be evaluated at any point in time $O_{\mathbb{F}}(\log |S|)$. An analogous statement holds if S is a coset of an additive subgroup.

[10] This follows from the general fact that, for $g(X,Y) := (f(X) - f(Y))/(X-Y)$, $g(X,X)$ is the formal derivative of $f(X)$. To see this, observe that $(X^n - Y^n)/(X-Y) = \sum_{i=0}^{n-1} X^i Y^{n-i-1}$. Setting $Y := X$ yields nX^{n-1}, the derivative of X^n.

If S is a coset in $\mathbb{F}$, an expression for $u_S(X, Y)$ can be found in time $O_{\mathbb{F}}(\log^2 |S|)$, and subsequently one can use this expression to evaluate $u_S(X, Y)$ at any point $(\alpha, \beta) \in \mathbb{F}^2$ in time $O_{\mathbb{F}}(\log |S|)$. [11]

Univariate sumcheck for cosets. For $S \subseteq \mathbb{F}, \hat{g} \in \mathbb{F}[X], \sigma \in \mathbb{F}$, define the polynomial:

$$\Sigma_S(\hat{g}, \sigma) := \begin{cases} X\hat{g}(X) + \sigma/|S| & \text{if } S \text{ is a multiplicative coset of } \mathbb{F} \\ \hat{g}(X) + \sigma X^{|S|-1} / \sum_{\alpha \in S} \alpha^{|S|-1} & \text{if } S \text{ is an additive coset of } \mathbb{F} \end{cases} .$$

Note that $\Sigma_S(\cdot\,,\cdot)$ may be viewed as an arithmetic circuit. We will use the following lemma from [BCRSVW19], which leads to a univariate analogue of the multivariate sumcheck protocol [LFKN92].

Lemma 3.4 ([BCRSVW19])**.** *Let S be a coset of $\mathbb{F}$, and let $\hat{f} \in \mathbb{F}[X]$ be such that $\deg(\hat{f}) < |S|$. Then $\sum_{\alpha \in S} \hat{f}(\alpha) = \sigma$ if and only if there exists $\hat{g}$ with $\deg(\hat{g}) < |S| - 1$ such that $\hat{f} \equiv \Sigma_S(\hat{g}, \sigma)$.*

[11] If S is a multiplicative coset in $\mathbb{F}$ then $u_S(X, Y) = (X^{|S|} - Y^{|S|})/(X - Y)$ and $u_S(X, X) = |S|X^{|S|-1}$, so both can be evaluated in time $O_{\mathbb{F}}(\log |S|)$. If S is an additive coset in $\mathbb{F}$ then $u_S(X, Y)$ is obtained directly from the linearized polynomial v_S, and $u_S(X, X)$ is the constant polynomial that equals the coefficient of the linear term in the linearized polynomial v_S.

4 Definition of holographic IOPs

A **holographic IOP** for an indexed relation $\mathcal{R}$ is specified by a tuple $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$, 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, given an index $\mathbb{i}$, the indexer $\mathbf{I}$ computes an encoding of $\mathbb{i}$, denoted $\mathbf{I}(\mathbb{i})$. Subsequently, in an online phase, the prover $\mathbf{P}$ receives as input a triple $(\mathbb{i}, \mathbb{x}, \mathbb{w})$, while the verifier $\mathbf{V}$ receives as input $\mathbb{x}$ and is granted oracle access to the encoded index $\mathbf{I}(\mathbb{i})$. The online phase consists of multiple rounds, and in each round the verifier $\mathbf{V}$ sends a message ρ_i and the prover $\mathbf{P}$ replies with a proof string Π_i, which the verifier can query at any location. At the end of the interaction, the verifier $\mathbf{V}$ accepts or rejects.

We say that HOL has perfect completeness and soundness error ϵ if the following holds.

- **Completeness.** For every index-instance-witness triple $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the probability that $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept in the interactive oracle protocol is 1.

- **Soundness.** For every index-instance pair $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R})$ and prover $\tilde{\mathbf{P}}$, the probability that $\tilde{\mathbf{P}}$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept in the interactive oracle protocol is at most ϵ.

The *round complexity* k is the number of back-and-forth message exchanges between the verifier and the prover. The *proof length* l is the sum of the length of the encoded index plus the lengths of all proof strings sent by the prover. The *query complexity* q is the total number of queries made by the verifier; this includes queries to the encoded index and to the oracles sent by the prover.

The holographic IOPs that we construct achieve the stronger property of *knowledge soundness* and optionally also *zero knowledge*. We define both of these properties below.

Knowledge soundness. HOL has knowledge error κ if there exists a probabilistic polynomial-time extractor $\mathbf{E}$ such that for every unbounded prover $\tilde{\mathbf{P}}$:

$$\Pr\left[\begin{array}{c} \exists\, j \text{ s.t. } (\mathbb{i}_j, \mathbb{x}_j, \mathbb{w}_j) \notin \mathcal{R} \\ \wedge \\ \langle \mathbf{P}_j, \mathbf{V}^{\mathbf{I}(\mathbb{i}_j)}(\mathbb{x}_j) \rangle = 1 \end{array} \;\middle|\; (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbf{P}}, \mathsf{aux}, \vec{\mathbb{w}}) \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}}(1^n) \right] \le \kappa$$

and, moreover, the following distributions are identical:

$$\left\{ (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbf{P}}, \mathsf{aux}) \;\middle|\; (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbf{P}}, \mathsf{aux}) \leftarrow \tilde{\mathbf{P}} \right\} \quad \text{and} \quad \left\{ (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbf{P}}, \mathsf{aux}) \;\middle|\; (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbf{P}}, \mathsf{aux}, \vec{\mathbb{w}}) \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}}(1^n) \right\} \ .$$

Zero knowledge. HOL has (perfect) zero knowledge with query bound b if there exists a probabilistic polynomial-time simulator $\mathbf{S}$ such that for every $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ and b-query algorithm $\tilde{\mathbf{V}}$ the random variables $\mathrm{View}(\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}), \tilde{\mathbf{V}})$ and $\mathbf{S}^{\tilde{\mathbf{V}}}(\mathbb{i}, \mathbb{x})$, defined below, are identical. (An algorithm is b-query if it makes *less than* b queries in total to any oracles it has access to.)

- $\mathrm{View}(\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}), \tilde{\mathbf{V}})$ is the *view* of $\tilde{\mathbf{V}}$, i.e., is the random variable $(r, a_1, \dots, a_q)$ where r is $\tilde{\mathbf{V}}$'s randomness and $a_1, \dots, a_q$ are the responses to $\tilde{\mathbf{V}}$'s queries determined by the oracles sent by $\mathbf{P}$.

- $\mathbf{S}^{\tilde{\mathbf{V}}}(\mathbb{i}, \mathbb{x})$ is the output of $\mathbf{S}(\mathbb{i}, \mathbb{x})$ when given straightline access to $\tilde{\mathbf{V}}$ ($\mathbf{S}$ may interact with $\tilde{\mathbf{V}}$, *without rewinding*, by exchanging messages with $\tilde{\mathbf{V}}$ and answering its oracle queries), *prepended* with $\tilde{\mathbf{V}}$'s randomness r. Note that r could be of super-polynomial size, so $\mathbf{S}$ cannot sample r on $\tilde{\mathbf{V}}$'s behalf and then output it; instead, we restrict $\mathbf{S}$ to not see r, and prepend r to $\mathbf{S}$'s output.

Public coins. HOL is *public-coin* if each verifier message to the prover is a random string. This means that the verifier's randomness is its messages $\rho_1, \dots, \rho_{\mathsf{k}} \in \{0,1\}^*$ and possibly additional randomness $\rho_{\mathsf{k}+1} \in \{0,1\}^*$ used after the interaction. All verifier queries can be postponed, without loss of generality, to a query phase that occurs after the interactive phase with the prover.

4.1 Reed–Solomon encoded holographic IOPs

Reed–Solomon encoded IOPs (RS-encoded IOPs) were introduced in [BCRSVW19] to provide a formal framework for separating protocol design from the technical issues introduced by low-degree testing. We adopt this formalism in this paper as well, with straightforward modifications for the holographic setting.

Informally, in an RS-encoded IOP, the prover and verifier engage in a public-coin IOP interaction and, after the interaction, the verifier outputs a set of algebraic claims about the prover's messages. The completeness condition requires that in the "yes" case, when the verifier interacts with the honest prover, the output claims are true with probability 1. The soundness condition requires that in the "no" case, when the verifier interacts with a malicious prover, at least one of the output claims will be false with high probability no matter what the prover's messages are. The holographic setting introduces the sole difference that the verifier's algebraic claims may include statements about the codewords output by the indexer.

In more detail, by "algebraic claim" we specifically mean a *rational constraint*, defined next.

Definition 4.1. *A* **rational constraint** *is a tuple* $\mathbf{c} = (p, q, d)$ *where* $p \colon \mathbb{F}^{1+\ell} \to \mathbb{F}$ *and* $q \colon \mathbb{F} \to \mathbb{F}$ *are arithmetic circuits, and* $d \in \mathbb{N}$ *is a degree bound. The arithmetic circuits* (p, q) *and a list of words* $f_1, \dots, f_\ell \colon L \to \mathbb{F}$ *jointly define the word* $(p,q)[f_1, \dots, f_\ell] \colon L \to \mathbb{F}$ *given by*

$$\forall\, a \in L\,,\ (p,q)[f_1, \dots, f_\ell](a) := \frac{p(a, f_1(a), \dots, f_\ell(a))}{q(a)} \enspace .$$

A rational constraint $\mathbf{c} = (p, q, d)$ *is* **satisfied with respect to** $f_1, \dots, f_\ell$ *if* $(p,q)[f_1, \dots, f_\ell] \in \mathrm{RS}[L, d]$.[12]

When describing rational constraints, we will often use the shorthand notation "$\deg(\hat{f}) \leq d$", where $f \colon L \to \mathbb{F}$ is defined as a rational equation over some oracles. This should be taken to mean the rational constraint $\mathbf{c} = (p, q, d)$ that is naturally induced by the expression that defines f.

A special type of rational constraint is a *boundary constraint*, defined next.

Definition 4.2. *A* **boundary constraint** *is a rational constraint that expresses a condition such as "*$\hat{f}(\alpha) = \beta$*" for some word* $f \colon L \to \mathbb{F}$ *and elements* $\alpha, \beta \in \mathbb{F}$*. Such a condition is represented via the rational constraint* $\mathbf{c} = (p, q, \deg(\hat{f}) - 1)$ *where* $p(X, Y) := Y - \beta$ *and* $q(X) := X - \alpha$*, which can be summarized as "*$\deg(\hat{g}) \leq \deg(\hat{f}) - 1$*" where* $g(a) := (f(a) - \beta)/(a - \alpha)$*. We denote this constraint simply by "*$\hat{f}(\alpha) = \beta$*".*

In the following we use $\mathrm{RS}[L, (d_1, \dots, d_k)] \subseteq (\mathbb{F}^k)^L$ to denote the interleaved Reed–Solomon code over L with degree bounds $(d_1, \dots, d_k)$, i.e., the set of $k \times |L|$ matrices where the i-th row is a codeword of $\mathrm{RS}[L, d_i]$ (which itself is all evaluations over L of univariate polynomials of degree at most d_i).

A *Reed–Solomon encoded holographic IOP* (RS-hIOP) for an indexed relation $\mathcal{R}$ is a tuple

$$(\mathbf{I}, \mathbf{P}, \mathbf{V}, \{\vec{d_{\mathbf{I}}}, \vec{d}_{\mathbf{P},1}, \dots, \vec{d}_{\mathbf{P},\mathsf{k}}\})$$

where $\mathbf{I}$ is a deterministic algorithm, $\mathbf{P}$ and $\mathbf{V}$ are probabilistic interactive algorithms, and $\vec{d_{\mathbf{I}}} \in \mathbb{N}^{\ell_0}$, $\vec{d}_{\mathbf{P},i} \in \mathbb{N}^{\ell_i}$ are vectors of degree bounds, that satisfies the following properties.

[12] For $a \in L$, if $q(a) = 0$ then we define $(p,q)[f_1, \dots, f_\ell](a) := \bot$. Note that if this holds for some $a \in L$ then, for *any* words $f_1, \dots, f_\ell$ and degree bound d, the rational constraint (p, q, d) is not satisfied by $f_1, \dots, f_\ell$.

Degree bounds: On input any $\mathbb{i}$, the indexer $\mathbf{I}$ outputs a codeword of $\mathrm{RS}[L, \vec{d}_{\mathbf{I}}]$. Moreover, on input any $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ and for every round i, the i-th message of $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ is a codeword of $\mathrm{RS}[L, \vec{d}_{\mathbf{P},i}]$.

Completeness: For every $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, all rational constraints output by $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ after interacting with $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ are satisfied with respect to $\mathbf{I}(\mathbb{i})$ and $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$'s messages with probability 1.

Soundness: For every $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R})$ and unbounded malicious prover $\tilde{\mathbf{P}}$ whose i-th message is a codeword of $\mathrm{RS}[L, \vec{d}_{\mathbf{P},i}]$, all rational constraints output by $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ after interacting with $\tilde{\mathbf{P}}$ are satisfied with respect to $\mathbf{I}(\mathbb{i})$ and the prover's messages with probability at most ϵ.

Often we will write that $\mathbf{V}$ "accepts", which means that all of the rational constraints it outputs are satisfied, or that it "rejects", which means that at least one rational constraint is not satisfied.

We conclude by discussing useful complexity measures for RS-encoded IOPs.

- The **query evaluation time** t_{q} is the natural complexity measure for $\mathbf{V}$, and equals the sum of the query evaluation times of the rational constraints output by $\mathbf{V}$. The query evaluation time of a single rational constraint $\mathbf{c} = (p, q, d)$ is the time required to compute $(p, q)[f_1, \ldots, f_\ell](a) \in \mathbb{F}$ given $a \in L$ and oracle access to $f_1, \ldots, f_\ell$ (and possibly additional information provided by the indexer). That is, it is the time needed to (construct and) evaluate the arithmetic circuits (p, q) at a single point.

- The **maximum degree** is a pair $(d_{\mathsf{c}}, d_{\mathsf{e}}) \in \mathbb{N} \times \mathbb{N}$ defined as follows.

  d_{c} is the "constraint degree", defined as the maximum specified degree of any oracle sent by the prover and any constraint output by the verifier, i.e., $d_{\mathsf{c}} := \max \vec{d}_{\mathbf{P},1} \cup \cdots \cup \vec{d}_{\mathbf{P},\mathsf{k}} \cup \{d : \mathbf{V} \text{ outputs } (p, q, d)\}$.

  d_{e} is the "effective degree", which is a quantity arising from the compilation from an RS-encoded IOP to a standard IOP via low-degree testing that is defined as follows:

  $$d_{\mathsf{e}} := \max \{d_{\mathsf{c}}\} \cup \left\{ \deg(p; \vec{d}_{\mathbf{I}}, \vec{d}_{\mathbf{P},1}, \ldots, \vec{d}_{\mathbf{P},\mathsf{k}}), d + \deg(q) : \mathbf{V} \text{ outputs } (p, q, d) \right\}$$

  where $\deg(P; \vec{d})$ for an arithmetic circuit $P \colon \mathbb{F}^{1+m} \to \mathbb{F}$ and degree bounds $d \in \mathbb{F}^m$ denotes the degree of the composed polynomial $P(X, Q_1(X), \ldots, Q_m(X))$ when $\deg(Q_i) = d_i$. Note that $d_{\mathsf{e}} \geq d_{\mathsf{c}}$.

4.2 Stronger notions of soundness

Aside from the standard notion of soundness above, there are two further soundness notions that arise when constructing non-interactive arguments from IOPs. These are round-by-round soundness [CCHLRR18; CMS19] and state-restoration soundness [BCS16], adapted to holographic IOPs. We discuss these below.

Round-by-round soundness. We begin by defining the notion of a (partial) transcript of an IOP, which means all verifier messages and proof strings up to a point where the prover is about to move

Definition 4.3. *A* **transcript** tr *of a holographic IOP* $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ *is a tuple of the form* $(\Pi_1, m_1, \ldots, \Pi_i, m_i)$ *for some* $i \in [\mathsf{k}]$*, where each* Π_j *is a prover (oracle) message and each* m_j *is a verifier message. We denote the empty transcript by* $\emptyset$*. A transcript is* **full** *if* $i = \mathsf{k}$*, where* k *is the round complexity of* $(\mathbf{I}, \mathbf{P}, \mathbf{V})$.

A protocol $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ has *round-by-round soundness error* ϵ_{rbr} if there exists a function State from the set of transcripts to $\{\mathsf{accept}, \mathsf{reject}\}$ such that for every transcript tr:

- if $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R})$ and $\mathsf{tr} = \emptyset$, then $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$;

- if $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$, then $\mathsf{rbr}(\mathsf{tr}) \leq \epsilon_{\mathrm{rbr}}$ where

$$\mathsf{rbr}(\mathsf{tr}) := \max_{\Pi} \Pr_{m} \left[\mathsf{State}(\mathbb{x}, \mathbb{i}, \mathsf{tr} \| \Pi \| m) = \mathsf{accept} \right] \quad ;$$

- if $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$ and tr is a full transcript, then $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x}; \mathsf{tr})$ rejects.

The notion of round-by-round soundness for *RS-encoded* holographic IOPs is as above, except that the maximum in the definition of rbr is taken over $\Pi_i \in \mathrm{RS}[L, \vec{d}_{\mathbf{P},i}]$, for tr a transcript of $i-1$ rounds, and the third condition above need only hold for full transcripts tr where $\Pi_i \in \mathrm{RS}[L, \vec{d}_{\mathbf{P},i}]$ for all i. In particular, $\mathsf{State}(\mathsf{tr})$ can be taken to be accept for any tr where the prover messages are not of the prescribed degrees.

State-restoration soundness. State-restoration soundness captures the ability of the prover to cheat when it is able to rewind the verifier a bounded number of times. State-restoration soundness essentially exactly captures the soundness of non-interactive arguments derived from IOPs via the BCS transform [BCS16]. In Section 10 we prove that continues to be true when we modify the BCS transform to construct *preprocessing* non-interactive arguments from *holographic* IOPs. However here we do not define state-restoration soundness because in our proof of the compiler we will rely on the BCS transform as a black box. We note only that if a protocol has round-by-round soundness error ϵ_{rbr} then it has state-restoration soundness error $\epsilon_{\mathrm{sr}}(t) \leq t \cdot \epsilon_{\mathrm{rbr}}$, where t is the bound on the number of rewinds. This fact is relevant because in Section 8 we will prove that the efficient holographic IOP for R1CS that we construct has small round-by-round soundness error.

Knowledge analogues. Each of the above notions of soundness induces a corresponding definition of knowledge soundness, whose details we omit for simplicity. See [CMS19] for the definition of round-by-round knowledge error, and see [BCS16] for the definition of state-restoration knowledge error.

5 Sumcheck for rational functions

We describe how to extend the univariate sumcheck protocol of [BCRSVW19] from univariate *polynomials* to univariate *rational functions*. We thus obtain a protocol for checking the value of the sum of a rational function $\hat{p}(X)/\hat{q}(X) \in \mathbb{F}(X)$ over a subgroup K of $\mathbb{F}$.

Definition 5.1. *Let $\mathcal{R}$ be the set of all pairs $(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, K, d_p, d_q, \sigma), (p, q)\big)$ such that $p \in \mathrm{RS}[L, d_p]$, $q \in \mathrm{RS}[L, d_q]$, and $\hat{q}(a) \neq 0$ for all $a \in K$. The promise relation $\mathcal{R}_{\mathrm{RSUM}} = (\mathcal{R}^{\mathrm{YES}}_{\mathrm{RSUM}}, \mathcal{R}^{\mathrm{NO}}_{\mathrm{RSUM}})$ is defined as follows: $\mathcal{R}^{\mathrm{YES}}_{\mathrm{RSUM}}$ is the subset of pairs in $\mathcal{R}$ such that $\sum_{a \in K} \hat{p}(a)/\hat{q}(a) = \sigma$, and $\mathcal{R}^{\mathrm{NO}}_{\mathrm{RSUM}} := \mathcal{R} \setminus \mathcal{R}^{\mathrm{YES}}_{\mathrm{RSUM}}$.*

Let $\sigma \in \mathbb{F}$ be the claimed value for the sum. We know from Lemma 3.4 that a polynomial $\hat{f}(X)$ of degree at most $|K| - 1$ sums to σ over K if and only if there exists a polynomial $\hat{g}(X)$ with degree at most $|K| - 2$ such that $\Sigma_K(\hat{g}, \sigma)(X)$ equals $\hat{f}(X)$. While we do not know how to obtain an equivalence like this one for the case of rational functions, there is a natural approach to build on the case of polynomials.

The prover computes the *polynomial* $\hat{f}(X)$ of minimal degree that agrees with the *rational function* $\hat{p}(X)/\hat{q}(X)$ on K, and then sends (the evaluation of) the corresponding polynomial $\hat{g}$ guaranteed by Lemma 3.4 (i.e., such that $\Sigma_K(\hat{g}, \sigma)(X) \equiv \hat{f}(X)$). The verifier can check that $\Sigma_K(\hat{g}, \sigma)(X)$ sums to σ via the rational constraint "$\deg(\hat{g}) \leq |K| - 2$". Then the verifier is left to check that $\Sigma_K(\hat{g}, \sigma)(X)$ agrees with $\hat{p}(X)/\hat{q}(X)$ on K, which is equivalent to checking that $\Sigma_K(\hat{g}, \sigma)(X)\hat{q}(X) - \hat{p}(X)$ vanishes on K (as $\hat{q}(a) \neq 0$ for all $a \in K$), which can be done via a standard use of a second rational constraint.

Construction 5.2 (rational sumcheck)**.** Let $(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, K, d_p, d_q, \sigma), (p, q)\big)$ be a pair in $\mathcal{R}$. In the rational sumcheck protocol, the honest prover $\mathbf{P}$ receives as input $(\mathbb{x}, \mathbb{w})$, and sends a codeword $g \in \mathrm{RS}[L, |K| - 2]$ that is obtained as follows: compute the unique polynomial $\hat{f}$ of degree at most $|K| - 1$ that agrees with $\hat{p}(X)/\hat{q}(X)$ on K; compute the unique polynomial $\hat{g}(X)$ of degree at most $|K| - 2$ such that $\Sigma_K(\hat{g}, \sigma)(X) \equiv \hat{f}(X)$; evaluate $\hat{g}(X)$ over L to obtain g. The honest verifier $\mathbf{V}$ receives as input $\mathbb{x}$, and outputs the following two rational constraints: "$\deg(\hat{g}) \leq |K| - 2$" and "$\deg(\hat{e}) \leq d_e$", where $e \colon L \to \mathbb{F}$ is a function and $d_e \in \mathbb{N}$ is a degree bound that are defined as follows:

$$\forall\, a \in L\,,\ e(a) := \frac{\Sigma_K(g, \sigma)(a) \cdot q(a) - p(a)}{v_K(a)} \quad \text{and} \quad d_e := \max(d_p, |K| - 1 + d_q) - |K| \ . \tag{1}$$

Formally, the above is an *RS-encoded PCP of proximity* for $\mathcal{R}_{\mathrm{RSUM}}$ (see [BCRSVW19] for definitions). For simplicity, in the lemma below we directly establish the properties that we need without this abstraction.

Lemma 5.3. *Let $(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, K, d_p, d_q, \sigma), (p, q)\big) \in \mathcal{R}$ be such that $L \cap K = \emptyset$, K is a subgroup of $\mathbb{F}$, and $\max(d_p, |K| - 1 + d_q) < |L|$. The protocol $(\mathbf{P}, \mathbf{V})$ in Construction 5.2 satisfies the following.*

1. Completeness: *if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}^{\mathrm{YES}}_{\mathrm{RSUM}}$ then $\mathbf{V}(\mathbb{x})$ outputs rational constraints that are satisfied by (p, q, g), where g is the oracle sent by the honest prover $\mathbf{P}(\mathbb{x}, \mathbb{w})$.*
2. Soundness: *if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}^{\mathrm{NO}}_{\mathrm{RSUM}}$ then for every malicious prover $\tilde{\mathbf{P}}$ at least one of the rational constraints output by $\mathbf{V}(\mathbb{x})$ is not satisfied by (p, q, g), where g is the oracle sent by the malicious prover $\tilde{\mathbf{P}}$.*

The protocol has constraint degree $\max(d_p - |K|, d_q - 1, |K| - 2)$ and effective degree $\max(d_p, |K| - 1 + d_q)$. The query evaluation time of the verifier is $O_{\mathbb{F}}(\log |K|)$.

Proof. We first argue completeness and then soundness.

Completeness. Suppose that $\sum_{a \in K} \hat{p}(a)/\hat{q}(a) = \sigma$. The honest prover $\mathbf{P}$ sends the polynomial $\hat{g}(X)$ with degree at most $|K| - 2$ such that $\Sigma_K(\hat{g}, \sigma)(X)$ agrees with $\hat{p}(X)/\hat{q}(X)$ on K; the existence of $\hat{g}(X)$ is guaranteed by Lemma 3.4. Since $\hat{q}(a) \neq 0$ for all $a \in K$, we also have that $\Sigma_K(\hat{g}, \sigma)(a) \cdot \hat{q}(a) = \hat{p}(a)$ for all

$a \in K$. Thus the polynomial $\Sigma_K(\hat{g}, \sigma)(X) \cdot \hat{q}(X) - \hat{p}(X)$ is divisible by the vanishing polynomial $v_K(X)$. We conclude that the two rational constraints "$\deg(\hat{g}) \leq |K| - 2$" and "$\deg(\hat{e}) \leq d_e$" are satisfied.

Soundness. Suppose that $\sum_{a \in K} \hat{p}(a)/\hat{q}(a) \neq \sigma$. Let g be the oracle sent by $\tilde{\mathbf{P}}$ and suppose that the rational constraint "$\deg(\hat{g}) \leq |K| - 2$" is satisfied (or else we are done). By Lemma 3.4 we know that $\sum_{a \in K} \Sigma_K(\hat{g}, \sigma)(a) = \sigma$. Hence there must exist $a^* \in K$ such that $\Sigma_K(\hat{g}, \sigma)(a^*) \cdot \hat{q}(a^*) \neq \hat{p}(a^*)$, so a^* is not a root of the polynomial $\Sigma_K(\hat{g}, \sigma)(X) \cdot \hat{q}(X) - \hat{p}(X)$. By definition of e, the polynomials $\Sigma_K(\hat{g}, \sigma)(X) \cdot \hat{q}(X) - \hat{p}(X)$ and $\hat{e}(X) \cdot v_K(X)$ agree on L. Since $d_e + |K| = \max(d_p, |K| - 1 + d_q) < |L|$, if the rational constraint "$\deg(\hat{e}) \leq d_e$" is also satisfied, then we can conclude that these two polynomials are identical, which is a contradiction because a^* is a root of $\hat{e}(X) \cdot v_K(X)$.

Efficiency. The verifier outputs a rational constraint on g, which is the oracle sent by the prover, and a rational constraint on e, which is the virtual oracle defined in Eq. (1). So the query evaluation time is dominated by the number of field operations to evaluate e at a single point, which is $O(\log |K|)$ (due to the need to evaluate the vanishing polynomial v_K at that point). The stated constraint and effective degrees can be obtained by keeping track of the degrees of the relevant real and virtual oracles in the protocol (as in the table) and then using the definitions in Section 4.1.

oracle	type	constraint degree	numerator degree	denominator degree
g	real	$\|K\| - 2$	–	–
e	virtual	d_e	$\max(d_p, \|K\| - 1 + d_q)$	$\|K\|$

□

Remark 5.4 (zero knowledge)**.** In Section 6, the above construction will be used as a subprotocol to evaluate the arithmetization of a *public* matrix. For this reason, we do not require any zero knowledge properties of the above construction (and indeed, the construction as described is not zero knowledge). Nonetheless, it is relatively straightforward to obtain a zero knowledge variant of this construction by using bounded independence and zero knowledge sumcheck as in [BCRSVW19].

6 Holographic lincheck

We describe a holographic variant of the lincheck protocol of [BCRSVW19]. The *lincheck problem* involves checking linear relations between encodings: given $H \subseteq \mathbb{F}$, Reed–Solomon codewords $f, g \in \mathrm{RS}[L, d]$, and matrix $M \in \mathbb{F}^{H \times H}$, check that $\hat{f}|_H = M \cdot \hat{g}|_H$. Below we consider matrices M that are given in a *sparse representation* (see Definition 3.1), as the protocol that we describe leverages this sparsity for efficiency.

Definition 6.1. *Let $\mathcal{R}$ be the set of all pairs $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, H, K, d, \langle M \rangle), 1^{\log |K|}, (f_1, f_2)\big)$ such that $\mathbb{F}$ is a finite field, L, H, K are subsets of $\mathbb{F}$, $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$ is a sparse representation of a matrix $M \in \mathbb{F}^{H \times H}$, $d \in \mathbb{N}$ is a degree bound, and $f_1, f_2 \in \mathrm{RS}[L, d]$ are codewords. The indexed promise relation $\mathcal{R}_{\mathrm{LIN}} = (\mathcal{R}_{\mathrm{LIN}}^{\mathrm{YES}}, \mathcal{R}_{\mathrm{LIN}}^{\mathrm{NO}})$ is such that $\mathcal{R}_{\mathrm{LIN}}^{\mathrm{YES}}$ is the subset of $\mathcal{R}$ with $\hat{f}_1|_H = M \cdot \hat{f}_2|_H$, and $\mathcal{R}_{\mathrm{LIN}}^{\mathrm{NO}} := \mathcal{R} \setminus \mathcal{R}_{\mathrm{LIN}}^{\mathrm{YES}}$.*

The goal of this section is to prove the following lemma.

Lemma 6.2. *Let $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, H, K, d, \langle M \rangle), 1^{\log |K|}, (f_1, f_2)\big) \in \mathcal{R}$ be such that H, K are subgroups of $\mathbb{F}$, $|L| \geq |K|$, and $L \cap (H \cup K) = \emptyset$. The protocol $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ in Construction 6.8 satisfies the following.*

1. Completeness: *if $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathrm{LIN}}^{\mathrm{YES}}$ then $\mathbf{V}(\mathbb{x})$ outputs rational constraints that are satisfied by (f, g) and the oracles sent by the honest prover $\mathbf{P}(\mathbb{i}, \mathbb{x})$.*

2. Soundness: *if $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathrm{LIN}}^{\mathrm{NO}}$ then for every malicious prover $\tilde{\mathbf{P}}$ all of the rational constraints output by $\mathbf{V}(\mathbb{x})$ are satisfied by (f_1, f_2) and the oracles sent by $\tilde{\mathbf{P}}$ with probability at most $2(|H| - 1)/(|\mathbb{F}| - |H|)$.*

Moreover, the construction can be made zero knowledge; the soundness error is then $2|H|/(|\mathbb{F}| - |H|)$.

The prover and indexer run in time $O_{\mathbb{F}}(|L| \log |L|)$, and the verifier's query evaluation time is $O_{\mathbb{F}}(\log |K|)$. The constraint degree is $\max(d-1, |H|-2, 2|K|-3)$ and the effective degree is $\max(|H|-1+d, 3|K|-3)$.

Formally, Construction 6.8 is an RS-encoded IOP of proximity for $\mathcal{R}_{\mathrm{LIN}}$. However, for simplicity, we directly establish the properties we need without this abstraction. The notion of zero knowledge is as usual for proximity notions: we require that if a malicious verifier $\tilde{\mathbf{V}}$ makes t queries across all of the oracles available to it, the simulator can reproduce its view by making t queries to *each of* the witness oracles.

The remainder of this section proceeds as follows. In Section 6.1, we describe a subprotocol for checking an evaluation of the low-degree (bivariate) extension of a matrix. In Section 6.2, we describe how to use this subprotocol to build a holographic lincheck protocol, proving Lemma 6.2. Throughout this section we rely on the notion of a sparse representation of a matrix (see Section 3.1) and on facts about vanishing polynomials and their derivatives (see Section 3.3).

6.1 Holographic proof for sparse matrix arithmetization

The *bivariate low-degree extension* of a given matrix $M \in \mathbb{F}^{H \times H}$ is the unique polynomial $\hat{M} \in \mathbb{F}[X, Y]$ of minimal degree such that $\hat{M}(a, b) = M_{a,b}$ for all $a, b \in H$. We wish to check statements of the form "$\hat{M}(\alpha, \beta) = \gamma$" for α, β chosen (almost) arbitrarily in $\mathbb{F}$.

Definition 6.3. *The indexed relation $\mathcal{R}_{\mathrm{MAT}}$ is the set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, H, K, \langle M \rangle), (\alpha, \beta, \gamma), \bot\big)$ where $\mathbb{F}$ is a finite field, H and K are subsets of $\mathbb{F}$, $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$ is a sparse representation of a matrix $M \in \mathbb{F}^{H \times H}$, $\alpha, \beta, \gamma \in \mathbb{F}$ are field elements, and $\hat{M}(\alpha, \beta) = \gamma$. (This relation has no witnesses.)*

The indexed relation $\mathcal{R}_{\mathrm{MAT}}$ is tractable: one can check if $\hat{M}(\alpha, \beta) = \gamma$ in time $O_{\mathbb{F}}(\|M\|)$ by directly computing the value of the low-degree extension $\hat{M}(X, Y)$ at (α, β). Without holography, it is not possible to verify this equation in time $o_{\mathbb{F}}(\|M\|)$ since in general $\hat{M}(\alpha, \beta)$ depends on every entry of M.

We show how to significantly reduce this cost via a protocol that holographically stores information about M in the encoded index in order to achieve an online verification time of $O(\log\|M\|)$. Our protocol relies on expressing the *bivariate* low-degree extension $\hat{M}(X,Y)$ in terms of *univariate* low-degree extensions that describe the non-zero entries of M. We explain this algebraic identity, and then how our protocol uses it.

Given a sparse representation $\langle M\rangle \colon K \to H \times H \times \mathbb{F}$, define $\hat{\mathsf{row}}_{\langle M\rangle}, \hat{\mathsf{col}}_{\langle M\rangle} \colon K \to H, \hat{\mathsf{val}}_{\langle M\rangle} \in \mathbb{F}[X]$ to be the unique polynomials of minimal degree such that for each $k \in K$, letting $(a,b,\alpha) := \langle M\rangle(k)$,

$$\hat{\mathsf{row}}_{\langle M\rangle}(k) := a \ , \quad \hat{\mathsf{col}}_{\langle M\rangle}(k) := b \ , \quad \hat{\mathsf{val}}_{\langle M\rangle}(k) := \frac{\alpha}{u_H(a,a)\cdot u_H(b,b)} \ .$$

The following claim expresses $\hat{M}$ in terms of $\hat{\mathsf{row}}_{\langle M\rangle}, \hat{\mathsf{col}}_{\langle M\rangle}, \hat{\mathsf{val}}_{\langle M\rangle}$.

Claim 6.4. *For any sparse representation $\langle M\rangle \colon K \to H \times H \times \mathbb{F}$ of a matrix $M \in \mathbb{F}^{H\times H}$,*

$$\hat{M}(X,Y) \equiv \sum_{k\in K} \frac{v_H(X)}{(X - \hat{\mathsf{row}}_{\langle M\rangle}(k))} \cdot \frac{v_H(Y)}{(Y - \hat{\mathsf{col}}_{\langle M\rangle}(k))} \cdot \hat{\mathsf{val}}_{\langle M\rangle}(k) \ .$$

Proof. Denote the right-hand side of the equation by $P(X,Y)$. Since $\hat{\mathsf{row}}_{\langle M\rangle}(k), \hat{\mathsf{col}}_{\langle M\rangle}(k) \in H$ for all $k \in K$, $P(X,Y)$ is a polynomial of degree at most $|H|-1$ in both X and Y. We now argue that $P(a,b) = M_{a,b}$ for arbitrary $a,b \in H$ (which implies that P agrees with $\hat{M}$ on $H\times H$ and hence that $P \equiv \hat{M}$). Suppose first that there is no $k \in K, \gamma \in \mathbb{F}$ such that $\langle M\rangle(k) = (a,b,\gamma)$. By definition of M, $M_{a,b} = 0$; moreover for any $k \in K$ either $v_H(X)/(X - \hat{\mathsf{row}}_{\langle M\rangle}(k))$ has a root at a or $v_H(Y)/(Y - \hat{\mathsf{col}}_{\langle M\rangle}(k))$ has a root at b, and so $P(a,b) = 0$ as well. Now suppose that there exists $k \in K, \gamma \in \mathbb{F}$ such that $\langle M\rangle(k) = (a,b,\gamma)$; note that k is unique because $\langle M\rangle$ is injective. Hence $P(a,b) = u_H(a,a)\cdot u_H(b,b)\cdot \hat{\mathsf{val}}_{\langle M\rangle}(k) = M_{a,b}$. □

Construction 6.5. The indexer $\mathbf{I}$ receives as input an index $\mathbb{i} = (\mathbb{F}, H, K, \langle M\rangle)$ along with an evaluation domain $L \subseteq \mathbb{F}$, computes the low-degree extensions $\hat{\mathsf{row}}_{\langle M\rangle}, \hat{\mathsf{col}}_{\langle M\rangle}, \hat{\mathsf{val}}_{\langle M\rangle}$, and then outputs their evaluations $\mathsf{row}_{\langle M\rangle}, \mathsf{col}_{\langle M\rangle}, \mathsf{val}_{\langle M\rangle} \in \mathrm{RS}[L, |K|-1]$. The indexer $\mathbf{I}$ also outputs descriptions of $\mathbb{F}, H, K$.

Subsequently, given an instance $\mathbb{x} = (\alpha, \beta, \gamma)$, the honest prover $\mathbf{P}$ receives as input $(\mathbb{i}, \mathbb{x})$ and the honest verifier $\mathbf{V}$ receives as input $\mathbb{x}$ and oracle access to $\mathbf{I}(\mathbb{i})$. The prover $\mathbf{P}$ and verifier $\mathbf{V}$ engage in the rational sumcheck protocol (see Section 5) to show that

$$\sum_{k\in K} \frac{v_H(\alpha)}{(\alpha - \hat{\mathsf{row}}_{\langle M\rangle}(k))} \cdot \frac{v_H(\beta)}{(\beta - \hat{\mathsf{col}}_{\langle M\rangle}(k))} \cdot \hat{\mathsf{val}}_{\langle M\rangle}(k) = \gamma \ .$$

In particular, the verifier $\mathbf{V}$ outputs the rational constraints "$\deg(\hat{g}) \le |K|-2$" for g sent by $\mathbf{P}$, and "$\deg(\hat{e}) \le 2|K|-3$" for $e \colon L \to \mathbb{F}$ defined as

$$\forall\, a \in L \,,\ e(a) := \frac{\Sigma_K(g,\gamma)(a)\cdot(\alpha - \mathsf{row}_{\langle M\rangle}(a))(\beta - \mathsf{col}_{\langle M\rangle}(a)) - v_H(\alpha)v_H(\beta)\mathsf{val}_{\langle M\rangle}(a)}{v_K(a)} \ . \qquad (2)$$

Lemma 6.6. *For any field $\mathbb{F}$ and evaluation domain $L \subseteq \mathbb{F}$, Construction 6.5 is an RS-encoded holographic PCP over domain L for the indexed relation $\mathcal{R}_{\mathrm{MAT}}$ with perfect completeness and perfect soundness, for indices $\mathbb{i} = (\mathbb{F}, H, K, \langle M\rangle)$ and instances $\mathbb{x} = (\alpha,\beta,\gamma)$ with H, K subgroups of $\mathbb{F}$, $|L| \ge |K|$, $L \cap K = \emptyset$, and $\alpha, \beta \in \mathbb{F} \setminus H$. In particular, the following properties hold.*

1. Completeness: *if $(\mathbb{i}, \mathbb{x}, \bot) \in \mathcal{R}_{\mathrm{MAT}}$ then $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ outputs rational constraints that are satisfied by the oracles sent by the honest prover $\mathbf{P}(\mathbb{i}, \mathbb{x})$.*
2. Soundness: *if $(\mathbb{i}, \mathbb{x}, \bot) \notin \mathcal{R}_{\mathrm{MAT}}$ then for every malicious prover $\tilde{\mathbf{P}}$ at least one of the rational constraints output by $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ is not satisfied by the oracles sent by $\tilde{\mathbf{P}}$.*

The constraint degree is $2|K|-3$*, and the effective degree is* $3|K|-3$*. The indexer and prover run in time* $O_{\mathbb{F}}(|L|\log|L|)$*, and the query evaluation time of the verifier is* $O_{\mathbb{F}}(\log|K|)$*.*

Proof. Completeness and soundness follow immediately from Claim 6.4, the completeness and soundness of the rational sumcheck protocol, and the observation that the denominator of the rational summand for $\hat{M}(\alpha,\beta)$ is nonzero for all $k\in K$ when $\alpha,\beta\in\mathbb{F}\setminus H$. The query evaluation time is dominated by the cost of evaluating v_H and v_K at a point, which is $O_{\mathbb{F}}(\log|H|+\log|K|)=O_{\mathbb{F}}(\log|K|)$. The constraint degree and effective degree are obtained from setting $d_p:=|K|-1$ and $d_q:=2|K|-2$ in the rational sumcheck protocol (see Lemma 5.3). □

6.2 The protocol

Recall from Section 3.3 that $\vec{r}:=\big(u_H(a,Y)\big)_{a\in H}\in\mathbb{F}[Y]^H$ is a vector of linearly independent polynomials in Y. The primary computational task in the lincheck protocol is to evaluate the low-degree extension $u_M(X,Y)$ of $\vec{r}M\in\mathbb{F}[Y]^H$ at a uniformly chosen point in $\mathbb{F}\times\mathbb{F}$. For this, we use the protocol for sparse matrix arithmetization discussed above, along with an observation showing that it suffices to compute the arithmetization of a matrix M^* related to M.

Claim 6.7. *For any matrix* $M\in\mathbb{F}^{H\times H}$*, let* $M^*\in\mathbb{F}^{H\times H}$ *be the matrix given by* $M^*_{a,b}:=M_{b,a}\cdot u_H(b,b)$ *for all* $a,b\in H$*; note that* $\|M^*\|=\|M\|$*. Then*

$$u_M(X,Y)\equiv\hat{M}^*(X,Y)\ .$$

Proof. By the definition of low-degree extension,

$$u_M(X,Y)\equiv\sum_{a\in H}(\vec{r}M)_a\cdot L_{a,H}(X)\equiv\sum_{a\in H}L_{a,H}(X)\sum_{b\in H}M_{b,a}\cdot u_H(b,Y)\ .$$

Recall that $u_H(b,Y)\equiv u_H(b,b)L_{b,H}(Y)$. Hence

$$u_M(X,Y)\equiv\sum_{a\in H}\sum_{b\in H}L_{a,H}(X)L_{b,H}(Y)M_{b,a}u_H(b,b)\equiv\hat{M}^*(X,Y)\ .$$ □

Construction 6.8 (holographic lincheck)**.** The indexer $\mathbf{I}$ receives as input an index $\mathbb{i}=(\mathbb{F},L,H,K,d,\langle M\rangle)$, computes a sparse representation $\langle M^*\rangle$ of the matrix M^* (as in Claim 6.7), and then runs the indexer of the sparse matrix arithmetization protocol (Construction 6.5) on the index $(\mathbb{F},H,K,\langle M^*\rangle)$; note that the output of the latter includes descriptions of $\mathbb{F},H,K$.

Subsequently, given an instance $\mathbb{x}=1^{\log|K|}$ and witness $\mathbb{w}=(f_1,f_2)$, the honest prover $\mathbf{P}$ receives as input $(\mathbb{i},\mathbb{x})$, the honest verifier $\mathbf{V}$ receives as input $\mathbb{x}$ and oracle access to $\mathbf{I}(\mathbb{i})$, and they engage in the following protocol.

1. $\mathbf{V}$ sends $\alpha\in\mathbb{F}\setminus H$ uniformly at random.
2. $\mathbf{P}$ sends the evaluation $t\in\mathrm{RS}[L,|H|-1]$ of the polynomial $\hat{t}(X):=u_M(X,\alpha)$.
3. $\mathbf{P},\mathbf{V}$ engage in the sumcheck protocol to show that $\sum_{b\in H}u_H(b,\alpha)\hat{f}_1(b)-\hat{t}(b)\hat{f}_2(b)=0$.
   That is, $\mathbf{P}$ sends $g_1\in\mathrm{RS}[L,|H|-2]$ and $\mathbf{V}$ outputs the rational constraints "$\deg(\hat{g}_1)\le|H|-2$" and "$\deg(\hat{h})\le d-1$" where

$$\forall\, b\in L\,,\ h(b):=\frac{u_H(b,\alpha)f_1(b)-t(b)f_2(b)-\Sigma_H(g_1,0)(b)}{v_H(b)}\ .$$

4. $\mathbf{V}$ sends $\beta \in \mathbb{F} \setminus H$ uniformly at random.
5. $\mathbf{P}$ sends the field element $\gamma := u_M(\beta, \alpha) = \hat{t}(\beta)$, and $\mathbf{V}$ outputs the boundary constraint "$\hat{t}(\beta) = \gamma$".
6. $\mathbf{P}, \mathbf{V}$ engage in the matrix arithmetization protocol (Construction 6.5) to show that $\hat{M}^*(\beta, \alpha) = \gamma$. That is, $\mathbf{P}$ sends $g_2 \in \mathrm{RS}[L, |K| - 2]$ and $\mathbf{V}$ outputs the rational constraints "$\deg(\hat{g}_2) \leq |K| - 2$" and "$\deg(\hat{e}) \leq 2|K| - 3$", where $e \colon L \to \mathbb{F}$ is as in Eq. (2) with g_2 in place of g and M^* in place of M.

Proof of Lemma 6.2. For $\alpha \in \mathbb{F}$, let $\vec{r}_\alpha := \big(u_H(b, \alpha)\big)_{b \in H} \in \mathbb{F}^H$. One can verify that $\vec{r}_\alpha M = \big(u_M(b, \alpha)\big)_{b \in H}$.

Completeness. Suppose that $\hat{f}_1|_H = M \cdot \hat{g}|_H$. Then for every $\alpha \in \mathbb{F} \setminus H$ it holds that

$$\sum_{b \in H} u_H(b, \alpha) \hat{f}_1(b) = \langle \vec{r}_\alpha, \hat{f}_1|_H \rangle = \langle \vec{r}_\alpha, M \cdot \hat{f}_2|_H \rangle = \langle \vec{r}_\alpha M, \hat{f}_2|_H \rangle = \sum_{b \in H} u_M(b, \alpha) \hat{f}_2(b) = \sum_{b \in H} \hat{t}(b) \hat{f}_2(b)$$

and so the sumcheck protocol in Step 3 succeeds. Next, Claim 6.7 tells us that $u_M(X, Y) \equiv \hat{M}^*(X, Y)$ and so for every $\beta \in \mathbb{F} \setminus H$ it holds that $u_M(\beta, \alpha) = \hat{M}^*(\beta, \alpha)$. This means that $\hat{M}^*(\beta, \alpha) = \hat{t}(\beta) = \gamma$, and so the matrix arithmetization subverifier accepts, and the boundary constraint "$\hat{t}(\beta) = \gamma$" is satisfied.

Soundness. Suppose that $\hat{f}_1|_H \neq M \cdot \hat{f}_2|_H$. Then $\Pr_{\alpha \in \mathbb{F} \setminus H}[\langle \vec{r}_\alpha, \hat{f}_1|_H \rangle = \langle \vec{r}_\alpha, M \cdot \hat{f}_2|_H \rangle] \leq (|H| - 1)/(|\mathbb{F}| - |H|)$. Suppose that this does not occur; then if $\hat{t}(X) \equiv u_M(X, \alpha)$, the sumcheck constraint is satisfied with probability zero, by the soundness of the sumcheck protocol. Hence we may assume $\hat{t}(X) \not\equiv u_M(X, \alpha)$, so $\Pr_{\beta \in \mathbb{F} \setminus H}[\hat{t}(\beta) = u_M(\beta, \alpha)] \leq (|H| - 1)/(|\mathbb{F}| - |H|)$. Thus assuming that the sumcheck constraint is satisfied, $\Pr_{\alpha, \beta \in \mathbb{F} \setminus H}[\hat{t}(\beta) = u_M(\beta, \alpha)] \leq 2(|H| - 1)/(|\mathbb{F}| - |H|)$. If this does not occur, and the boundary constraint "$\hat{t}(\beta) = \gamma$" is satisfied, then $\gamma \neq u_H(\beta, \alpha) = \hat{M}^*(\beta, \alpha)$, and so the matrix arithmetization subverifier outputs a rational constraint that is not satisfied.

Zero knowledge. Note that since M is part of the index rather than the witness, it is not relevant for zero knowledge; in particular, the simulator has access to M. Hence to ensure zero knowledge we need only modify Step 3 to use the zero knowledge sumcheck protocol. This adds an additional oracle (the random mask) but not an additional round, since we can run the protocols in parallel. The soundness error increases by $1/|\mathbb{F}|$. The sumcheck simulator satisfies the property that the view of a malicious verifier making t queries across all oracles can be simulated by making t queries to the summand. Since the summand is defined pointwise with respect to (f_1, f_2), this results in at most t queries to each of f_1, f_2.

Efficiency. The indexer runs the indexer of the holographic protocol for sparse matrix arithmetization on (a modification of) the given matrix, and so runs in time $O_{\mathbb{F}}(|L| \log |L|)$. The first message of the prover is for the sumcheck protocol and the second message of the prover is for the sparse matrix arithmetization protocol, and so the prover runs in time $O_{\mathbb{F}}(|L| \log |L|)$. The query evaluation time of the verifier is dominated by the cost to evaluate the vanishing polynomials v_H and v_K at a point, and so is $O_{\mathbb{F}}(\log |H| + \log |K|) = O_{\mathbb{F}}(\log |K|)$.

The constraint degree is $\max(d - 1, |H| - 2, 2|K| - 3)$ and the effective degree is $\max(|H| - 1 + d, 3|K| - 3)$, as can be seen by keeping track of the degrees of all relevant real and virtual oracles in the protocol (as in the table) and then using the definitions in Section 4.1.

oracle	type	constraint degree	numerator degree	denominator degree
g_1	real	$\lvert H\rvert - 2$	–	–
h	virtual	$d - 1$	$\lvert H\rvert - 1 + d$	$\lvert H\rvert$
g_2	real	$\lvert K\rvert - 2$	–	–
e	virtual	$2\lvert K\rvert - 3$	$3\lvert K\rvert - 3$	$\lvert K\rvert$

□

7 RS-encoded holographic IOP for R1CS

We describe an RS-encoded holographic IOP for R1CS. The main subroutine that we use is the RS-encoded holographic protocol for lincheck that we obtained in Section 6.

Theorem 7.1. *Fix some $L \subseteq \mathbb{F}$ and $\mathsf{b} \in \mathbb{N}$. Construction 7.2 below is a RS-encoded holographic IOP of knowledge for $\mathcal{R}_{\text{R1CS}}$ (Definition 3.2) over domain L for indices $(\mathbb{F}, H, K, \langle A\rangle, \langle B\rangle, \langle C\rangle)$ such that H, K are subgroups of $\mathbb{F}$, $|L| \geq |K|$, and $L \cap (H \cup K) = \emptyset$. The protocol has 5 messages (prover moves first), is zero knowledge against verifiers making less than b queries, and has soundness error $|H|/(|\mathbb{F}| - |H|)$. The index length is $O_{\mathbb{F}}(|L|)$, and the proof length is $O_{\mathbb{F}}(|L|)$. The prover and indexer run in time $O_{\mathbb{F}}(|L| \log |L|)$ and the verifier runs in time $O_{\mathbb{F}}(|x| + \log |K|)$. The constraint degree is $\max(|H| + \mathsf{b} - 2, 2|K| - 3)$, and the effective degree is $\max(2|H| + \mathsf{b} - 2, 3|K| - 3)$.*

Construction 7.2. We describe an RS-encoded holographic IOP $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ for R1CS. (See Fig. 4 for a diagram of this protocol after applying optimizations described in Remark 7.3 below.) In the description below we denote by $(\mathbf{I}_{\text{LIN}}, \mathbf{P}_{\text{LIN}}, \mathbf{V}_{\text{LIN}})$ the zero knowledge holographic protocol for lincheck (Construction 6.8).

The indexer $\mathbf{I}$ receives as input an index $\mathbb{i} = (\mathbb{F}, H, K, \langle A\rangle, \langle B\rangle, \langle C\rangle)$, computes the encoded index $\mathbb{I}_M \leftarrow \mathbf{I}_{\text{LIN}}(\mathbb{i}_{\text{LIN}}^M)$ where $\mathbb{i}_{\text{LIN}}^M := (\mathbb{F}, L, H, K, |H| + \mathsf{b} - 1, \langle M\rangle)$ for each $M \in \{A, B, C\}$ for each $M \in \{A, B, C\}$, and then outputs the tuple $(\mathbb{I}_A, \mathbb{I}_B, \mathbb{I}_C)$. (Implicitly this includes descriptions of $\mathbb{F}, H, K$; recall also that in an RS-encoded protocol all parties have access to a description of the evaluation domain L.)

Subsequently, the prover $\mathbf{P}$ receives as input the index $\mathbb{i}$, an instance $\mathbb{x} = (I, x)$, and a witness $\mathbb{w} = w$; the verifier $\mathbf{V}$ receives as input the instance $\mathbb{x}$ only. Let $z := (x, w) \in \mathbb{F}^H$ be the full variable assignment.

1. **Compute LDE of the input.** Before the interaction, the prover $\mathbf{P}$ constructs $\hat{f}_x(X)$, the unique polynomial of degree less than $|I|$ such that, for all $b \in I$, $\hat{f}_x(b) = x_b$. Define $f_x := \hat{f}_x|_L$. Note that the verifier $\mathbf{V}$, which knows x, can evaluate $\hat{f}_x(X)$ at any point in $\mathbb{F}$ in time $O_{\mathbb{F}}(|x|)$.
2. **Witness and auxiliary oracles.** The prover $\mathbf{P}$ sends to the verifier $\mathbf{V}$ the oracles

$$f_w \in \mathrm{RS}[L, |H| - |I| + \mathsf{b} - 1] \quad \text{and} \quad f_{Az}, f_{Bz}, f_{Cz} \in \mathrm{RS}[L, |H| + \mathsf{b} - 1]$$

defined as follows.

- $f_w := \bar{f}_w|_L$ where $\bar{f}_w$ is a random polynomial of degree less than $|H| - |I| + \mathsf{b}$ such that

$$\forall\, a \in H \setminus I, \quad \bar{f}_w(a) = \frac{w_a - \hat{f}_x(a)}{v_I(a)} \quad .$$

- $f_{Az} := \bar{f}_{Az}|_L$ where $\bar{f}_{Az}$ is a random polynomial of degree less than $|H| + \mathsf{b}$ such that, for all $a \in H$, $\bar{f}_{Az}(a) = \sum_{b \in H} A_{a,b} \cdot z_b = (Az)_a$. The other codewords, f_{Bz} and f_{Cz}, are defined similarly.

The codewords f_x and f_w implicitly define the "virtual oracle" $f_z \in \mathrm{RS}[L, |H| + \mathsf{b} - 1]$ where $f_z(a) := f_w(a)v_I(a) + f_x(a)$ for $a \in L$. Note that $\hat{f}_z(a) = z_a$ for all $a \in H$, so $\hat{f}_z$ is a low-degree extension of z.

3. **Rowcheck.** To test that $\hat{f}_{Az}|_H \circ \hat{f}_{Bz}|_H = \hat{f}_{Cz}|_H$, the verifier $\mathbf{V}$ outputs the rational constraint "$\deg(\hat{s}) \leq |H| + 2\mathsf{b} - 2$" where $s \colon L \to \mathbb{F}$ is defined as

$$\forall\, a \in L\,, \; s(a) := \frac{f_{Az}(a) \cdot f_{Bz}(a) - f_{Cz}(a)}{v_H(a)} \quad .$$

4. **Linchecks.** To test that $\hat{f}_{Mz}|_H = M \cdot \hat{f}_z|_H$ for each $M \in \{A, B, C\}$, the prover $\mathbf{P}$ and verifier $\mathbf{V}$ run the following in parallel. Recall that the verifier $\mathbf{V}$ has oracle access to the encoded index is $(\mathbb{I}_A, \mathbb{I}_B, \mathbb{I}_C)$.

(a) $\big(\mathbf{P}_{\text{LIN}}(\mathbb{i}_{\text{LIN}}^{A}, 1^{\log|K|}, (f_{Az}, f_z)), \mathbf{V}_{\text{LIN}}^{f_{Az}, f_z, \mathbb{I}_A}(1^{\log|K|})\big)$.
(b) $\big(\mathbf{P}_{\text{LIN}}(\mathbb{i}_{\text{LIN}}^{B}, 1^{\log|K|}, (f_{Bz}, f_z)), \mathbf{V}_{\text{LIN}}^{f_{Bz}, f_z, \mathbb{I}_B}(1^{\log|K|})\big)$.
(c) $\big(\mathbf{P}_{\text{LIN}}(\mathbb{i}_{\text{LIN}}^{C}, 1^{\log|K|}, (f_{Cz}, f_z)), \mathbf{V}_{\text{LIN}}^{f_{Cz}, f_z, \mathbb{I}_C}(1^{\log|K|})\big)$.

Proof. Fix an index $\mathbb{i} = (\mathbb{F}, H, K, \langle A\rangle, \langle B\rangle, \langle C\rangle)$ and instance $\mathbb{x} = (I, x)$.

Completeness. Suppose that $(\mathbb{i}, \mathbb{x}, w) \in \mathcal{R}_{\text{R1CS}}$, and let $z := (x, w)$. Note that, by construction, $\hat{f}_z$ is a low-degree extension of z. Since $Az \circ Bz = Cz$, we know that $\hat{f}_{Az}|_H \circ \hat{f}_{Bz}|_H = \hat{f}_{Cz}|_H$ and $\hat{f}_{Mz}|_H = M \cdot \hat{f}_z|_H$ for each $M \in \{A, B, C\}$. Hence, for all $a \in H$ it holds that $f_{Az}(a) \cdot f_{Bz}(a) - f_{Cz}(a) = 0$, and so $\hat{s}(X)$ is a polynomial of degree at most $2(|H| + \mathsf{b} - 1) - |H| = |H| + 2\mathsf{b} - 2$, so the rational constraint in Step 3 is satisfied. Moreover, the holographic lincheck protocol in Step 4a yields rational constraints which are satisfied; by a similar argument, Steps 4b and 4c yield satisfied rational constraints.

Soundness. Suppose that $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{\text{R1CS}})$. Let f_z be the virtual oracle induced by f_w as sent by $\tilde{\mathbf{P}}$. Then either $\hat{f}_{Az}|_H \circ \hat{f}_{Bz}|_H \neq \hat{f}_{Cz}|_H$, or there exists $M \in \{A, B, C\}$ such that $\hat{f}_{Mz}|_H \neq M \cdot \hat{f}_z|_H$. In the former case, the rational constraint output in Step 3 is not satisfied (it is satisfied with probability 0). In the latter case, by the soundness of the holographic lincheck protocol the probability that all of the rational constraints output by the verifier in Steps 4a, 4b and 4c are satisfied is at most $|H|/(|\mathbb{F}| - |H|)$. The soundness error is given by maximizing over these two cases.

Proof of knowledge. The extractor $\mathbf{E}$ operates as follows: run $\tilde{\mathbf{P}}$ to obtain $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\mathbf{P}}, \mathsf{aux})$. For each $\mathbf{P}_j$, run it until it outputs f_w and output $\hat{f}_w|_{H\setminus I}$. From the soundness analysis, it holds that if extraction fails to produce a valid witness, the verifier accepts with probability at most $\epsilon := |H|/(|\mathbb{F}| - |H|)$. Let $S := \{f_w : (\mathbb{i}_j, \mathbb{x}_j, \hat{f}_w|_{H\setminus I}) \in \mathcal{R}_{\text{R1CS}}\}$. Then

$$\begin{aligned}\Pr[\mathbf{V}\text{ accepts}] &= \Pr[\mathbf{V}\text{ accepts}|f_w \in S]\Pr[f_w \in S] + \Pr[\mathbf{V}\text{ accepts}|f_w \notin S]\Pr[f_w \notin S] \\ &\leq \Pr[\mathbf{E}\text{ succeeds}] + \epsilon \cdot \Pr[\mathbf{E}\text{ fails}] = (1-\epsilon)\Pr[\mathbf{E}\text{ succeeds}] + \epsilon\end{aligned}$$

And so if $\mathbf{V}$ accepts with probability μ, then $\mathbf{E}$ succeeds with probability at least $(\mu - \epsilon)/(1 - \epsilon)$. Taking a union bound over all j yields a knowledge error of $k \cdot (\mu - \epsilon)/(1 - \epsilon)$ when $\tilde{\mathbf{P}}$ outputs k instances.

Zero knowledge. The simulator $\mathbf{S}$ simulates the oracles $f_w, f_{Az}, f_{Bz}, f_{Cz}$ by answering $\tilde{\mathbf{V}}$'s queries with uniformly random elements of $\mathbb{F}$. It runs the simulator for the zero knowledge holographic lincheck protocol as appropriate, answering the subsimulators' queries to the oracles with uniformly random field elements. Since $\tilde{\mathbf{V}}$ makes $t < \mathsf{b}$ queries across all oracles, the guarantees of the subsimulators ensure that we only need to simulate at most t evaluations of each of $f_w, f_{Az}, f_{Bz}, f_{Cz}$ in L (with $L \cap H = \emptyset$), which by bounded independence properties of random polynomials will be uniformly random elements of $\mathbb{F}$. For a detailed simulator construction for a similar protocol, see [BCRSVW19, Section 7.1].

Efficiency. The running time of the indexer follows from the running time of the lincheck indexer; in particular, its computation cost is dominated by the cost of a constant number of FFTs over L. The running time of the prover is similarly dominated. The constraint cost of the verifier consists of evaluating the low degree extension of x at a single point in $\mathbb{F}$, and running the lincheck and rowcheck subverifiers whose cost is dominated by evaluating v_H and v_K; using preprocessing this can be achieved for H, K subgroups of $\mathbb{F}$ in time $O_{\mathbb{F}}(\log|K|)$. □

Remark 7.3 (batching linchecks)**.** We would like to batch the three lincheck protocols from Steps 4a to 4c into a single one, similarly to what is done in the non-holographic protocol for R1CS of [BCRSVW19]. Informally, we want the verifier to send random elements η_A, η_B, η_C and then run a holographic lincheck for the matrix $\eta_A A + \eta_B B + \eta_C C$. However doing this requires some care because the verifier only has access to the encoded indices $(\mathbb{I}_A, \mathbb{I}_B, \mathbb{I}_C)$ for the matrices A, B, C, as opposed to an encoded index for the matrix $\eta_A A + \eta_B B + \eta_C C$, and our holographic lincheck protocol is *not* linear in the encoded indices.

We now explain how to overcome this issue by "opening up" the lincheck protocol into its components, described in Construction 6.8 in Section 6. The resulting protocol is summarized in Fig. 4.

The first message of the verifier consists of random elements η_A, η_B, η_C for the random linear combination, along with the random challenge α prescribed by Step 1 of the holographic lincheck protocol.

The subsequent two steps are straightforward to adapt due to linearity:

- In Step 2, the prover must send the evaluation of the polynomial $\hat{t}(X) := u_{\eta_A A+\eta_B B+\eta_C C}(X, \alpha)$, which by linearity equals to $\sum_{M\in\{A,B,C\}} \eta_M u_M(X, \alpha)$. The prover thus sends $t := \hat{t}\ |_L \in \mathrm{RS}[L, |H| - 1]$.
- In Step 3, the prover and verifier must run the zero knowledge sumcheck protocol (relative to a random mask r sent earlier) to show that $\sum_{b\in H} u_H(b, \alpha)\hat{f}_\$(b) - \hat{t}(b)\hat{f}_z(b) = 0$ where $\hat{f}_\$(X)$ is the low-degree extension of the vector $(\sum_{M\in\{A,B,C\}} \eta_M M)z$. By linearity, $\hat{f}_\$(X) = \sum_{M\in\{A,B,C\}} \eta_M \hat{f}_M(X)$, which means that the verifier can do this since the prover has sent f_z, f_A, f_B, f_C.

Then there are two steps that remain unchanged: the verifier sends the random challenge β prescribed by Step 4 of the holographic lincheck protocol, and the prover answers with the evaluation $\gamma := \hat{t}(\beta)$ prescribed in Step 5 of the holographic lincheck protocol.

The final step of the holographic lincheck protocol, Step 6, involves a rational sumcheck checking that γ is the value of the low-degree extension of $\sum_{M\in\{A,B,C\}} \eta_M M$ at (β, α). This is the step that lacks linear structure and we need to modify it. Specifically we need to turn the expression

$$\sum_{M\in\{A,B,C\}} \eta_M \frac{v_H(\alpha)}{(\alpha - \hat{\mathsf{row}}_{\langle M^*\rangle}(X))} \cdot \frac{v_H(\beta)}{(\beta - \hat{\mathsf{col}}_{\langle M^*\rangle}(X))} \cdot \hat{\mathsf{val}}_{\langle M^*\rangle}(X)$$

into a rational function in X. This is achieved by "multiplying up" denominators, to obtain the rational function $\hat{p}(X)/\hat{q}(X)$ where

$$\hat{p}(X) := v_H(\alpha)v_H(\beta) \sum_{M\in\{A,B,C\}} \eta_M \hat{\mathsf{val}}_{\langle M^*\rangle}(X) \prod_{N\in\{A,B,C\}\setminus\{M\}} (\alpha - \hat{\mathsf{row}}_{\langle N^*\rangle}(X))(\beta - \hat{\mathsf{col}}_{\langle N^*\rangle}(X)) \tag{3}$$

$$\hat{q}(X) := \prod_{M\in\{A,B,C\}} (\alpha - \hat{\mathsf{row}}_{\langle M^*\rangle}(X))(\beta - \hat{\mathsf{col}}_{\langle M^*\rangle}(X)) \ . \tag{4}$$

Crucially, the verifier can easily evaluate $\hat{p}$ and $\hat{q}$ at any point on L by having oracle access to the encoded indices $(\mathbb{I}_A, \mathbb{I}_B, \mathbb{I}_C)$.

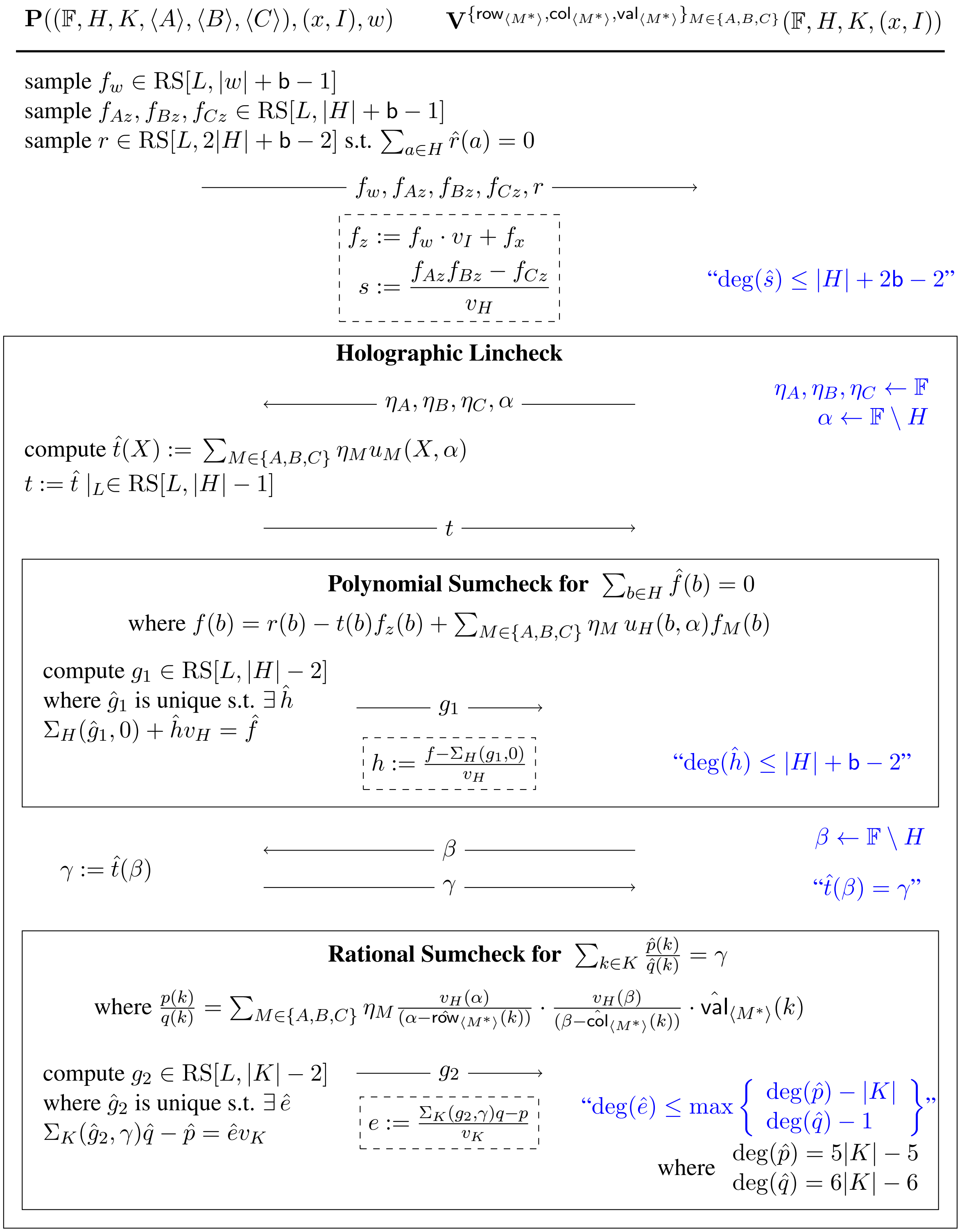
$\mathbf{P}((\mathbb{F}, H, K, \langle A \rangle, \langle B \rangle, \langle C \rangle), (x, I), w)$
$\mathbf{V}^{\{\mathsf{row}_{\langle M^* \rangle}, \mathsf{col}_{\langle M^* \rangle}, \mathsf{val}_{\langle M^* \rangle}\}_{M \in \{A,B,C\}}}(\mathbb{F}, H, K, (x, I))$
sample $f_w \in \mathrm{RS}[L, |w| + \mathsf{b} - 1]$
sample $f_{Az}, f_{Bz}, f_{Cz} \in \mathrm{RS}[L, |H| + \mathsf{b} - 1]$
sample $r \in \mathrm{RS}[L, 2|H| + \mathsf{b} - 2]$ s.t. $\sum_{a \in H} \hat{r}(a) = 0$
$\xrightarrow{f_w, f_{Az}, f_{Bz}, f_{Cz}, r}$
$f_z := f_w \cdot v_I + f_x$
$s := \frac{f_{Az} f_{Bz} - f_{Cz}}{v_H}$
"$\deg(\hat{s}) \leq |H| + 2\mathsf{b} - 2$"
Holographic Lincheck
$\xleftarrow{\eta_A, \eta_B, \eta_C, \alpha}$
$\eta_A, \eta_B, \eta_C \leftarrow \mathbb{F}$
$\alpha \leftarrow \mathbb{F} \setminus H$
compute $\hat{t}(X) := \sum_{M \in \{A,B,C\}} \eta_M u_M(X, \alpha)$
$t := \hat{t} \mid_L \in \mathrm{RS}[L, |H| - 1]$
$\xrightarrow{t}$
Polynomial Sumcheck for $\sum_{b \in H} \hat{f}(b) = 0$
where $f(b) = r(b) - t(b) f_z(b) + \sum_{M \in \{A,B,C\}} \eta_M \, u_H(b, \alpha) f_M(b)$
compute $g_1 \in \mathrm{RS}[L, |H| - 2]$
where $\hat{g}_1$ is unique s.t. $\exists \, \hat{h}$
$\Sigma_H(\hat{g}_1, 0) + \hat{h} v_H = \hat{f}$
$\xrightarrow{g_1}$
$h := \frac{f - \Sigma_H(g_1, 0)}{v_H}$
"$\deg(\hat{h}) \leq |H| + \mathsf{b} - 2$"
$\gamma := \hat{t}(\beta)$
$\xleftarrow{\beta}$
$\xrightarrow{\gamma}$
$\beta \leftarrow \mathbb{F} \setminus H$
"$\hat{t}(\beta) = \gamma$"
Rational Sumcheck for $\sum_{k \in K} \frac{\hat{p}(k)}{\hat{q}(k)} = \gamma$
where $\frac{p(k)}{q(k)} = \sum_{M \in \{A,B,C\}} \eta_M \frac{v_H(\alpha)}{(\alpha - \hat{\mathsf{row}}_{\langle M^* \rangle}(k))} \cdot \frac{v_H(\beta)}{(\beta - \hat{\mathsf{col}}_{\langle M^* \rangle}(k))} \cdot \hat{\mathsf{val}}_{\langle M^* \rangle}(k)$
compute $g_2 \in \mathrm{RS}[L, |K| - 2]$
where $\hat{g}_2$ is unique s.t. $\exists \, \hat{e}$
$\Sigma_K(\hat{g}_2, \gamma) \hat{q} - \hat{p} = \hat{e} v_K$
$\xrightarrow{g_2}$
$e := \frac{\Sigma_K(g_2, \gamma) q - p}{v_K}$
"$\deg(\hat{e}) \leq \max \left\{ \begin{array}{c} \deg(\hat{p}) - |K| \\ \deg(\hat{q}) - 1 \end{array} \right\}$"
where $\deg(\hat{p}) = 5|K| - 5$
$\deg(\hat{q}) = 6|K| - 6$

Figure 4: Diagram of our RS-encoded holographic IOP for R1CS (Construction 7.2), after applying the optimizations described in Remark 7.3 (which batch the three holographic linchecks into one holographic protocol).

8 Holographic IOP for R1CS

We construct an efficient holographic IOP for rank-1 constraint satisfiability (R1CS). Our preprocessing zkSNARK is obtained by applying our compiler to this protocol (reparametrized to reduce soundness error).

Theorem 8.1. *There exists a public-coin holographic IOP* $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ *for the indexed relation* $\mathcal{R}_{\mathrm{R1CS}}$ *(Definition 3.2) that is a zero knowledge proof of knowledge with the following efficiency features.*

- Indexing. *The indexer* $\mathbf{I}$*, given an index* $\mathbb{i} = (\mathbb{F}, H, K, \langle A \rangle, \langle B \rangle, \langle C \rangle)$*, where* H, K *are subgroups of* $\mathbb{F}$*, runs in time* $O_{\mathbb{F}}(|K| \log |K|)$ *to compute an encoded index* $\mathbf{I}(\mathbb{i})$ *of size* $O_{\mathbb{F}}(|K|)$*. Note that* $|\mathbf{I}(\mathbb{i})| = O(|\mathbb{i}|)$*.*

- Proving and verification. *To achieve zero knowledge against* b *queries, the prover* $\mathbf{P}$ *and verifier* $\mathbf{V}$ *interact over* $O(\log(|K| + \mathsf{b}))$ *rounds with a round-by-round soundness (and knowledge) error of* $O((|K| + \mathsf{b})/|\mathbb{F}| + \epsilon_{\mathrm{FRI}}(\mathbb{F}, \rho, \delta))$ *where* $\rho, \delta = \Theta(1)$*. The prover* $\mathbf{P}$ *runs in time* $O_{\mathbb{F}}((|K| + \mathsf{b}) \log(|K| + \mathsf{b}))$*, and the total length of the proof oracles that it outputs is* $O_{\mathbb{F}}(|K| + \mathsf{b})$*. The verifier* $\mathbf{V}$ *runs in time* $O_{\mathbb{F}}(|x| + \log(|K| + \mathsf{b}))$*, and makes* $O(\log(|K| + \mathsf{b}))$ *queries to the encoded index and proof oracles.*

Above, $\epsilon_{\mathrm{FRI}}(\mathbb{F}, \rho, \delta)$ denotes the round-by-round soundness error of the FRI low-degree test [BBHR18] over the field $\mathbb{F}$ for proximity parameter δ and rate parameter ρ.

The rest of this section is organized as follows: (1) we introduce a generic theorem (Theorem 8.2) that allows us to "compile" an RS-encoded holographic IOP into a holographic IOP via a low-degree test; then (2) we show how to apply this theorem with the FRI low-degree test [BBHR18] to prove Theorem 8.1. The generic theorem, stated next, is adapted from [BCRSVW19] to handle holography and round-by-round soundness, and to more carefully account for the running time of the verifier.

Theorem 8.2 (adapted from [BCRSVW19])**.** *Suppose that we are given:*

- *an RS-encoded holographic IOP* $\mathsf{HOL}_{\mathcal{R}} = (\mathbf{I}_{\mathcal{R}}, \mathbf{P}_{\mathcal{R}}, \mathbf{V}_{\mathcal{R}}, \{\vec{d}_{\mathbf{I}}, \vec{d}_{\mathbf{P},1}, \ldots, \vec{d}_{\mathbf{P},\mathsf{k}}\})$ *over* L*, with maximum degree* $(d_{\mathsf{c}}, d_{\mathsf{e}})$*, for an indexed relation* $\mathcal{R}$*;*
- *a low-degree test* $(\mathbf{P}_{\mathrm{LDT}}, \mathbf{V}_{\mathrm{LDT}})$ *for the Reed–Solomon code* $\mathrm{RS}[L, d_{\mathsf{c}}]$.

Fix any proximity parameter δ^{LDT} *such that*

$$\delta^{\mathrm{LDT}} < \min\left(\frac{1 - 2\rho_c}{2}, \frac{1 - \rho_c}{3}, 1 - \rho_e\right) \quad \textit{where} \quad \rho_c := \frac{d_{\mathsf{c}} + 1}{|L|} \quad \textit{and} \quad \rho_e := \frac{d_{\mathsf{e}} + 1}{|L|} \ .$$

Then we can combine the above two ingredients to obtain a holographic IOP $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ *for* $\mathcal{R}$ *with* $\mathsf{k}^{\mathcal{R}} + \mathsf{k}^{\mathrm{LDT}}$ *rounds, query complexity* $\mathsf{q}_{\pi}^{\mathrm{LDT}} + \mathsf{q}_{\mathrm{w}}^{\mathrm{LDT}} \cdot (\mathsf{k}^{\mathcal{R}} + 1)$*, proof length* $\mathsf{l}^{\mathcal{R}} + \mathsf{l}^{\mathrm{LDT}}$*, soundness error* $\epsilon^{\mathcal{R}} + \epsilon^{\mathrm{LDT}} + |L|/|\mathbb{F}|$*, and* **round-by-round soundness error** $\max(\epsilon_{\mathrm{rbr}}^{\mathcal{R}}, \epsilon_{\mathrm{rbr}}^{\mathrm{LDT}}, |L|/|\mathbb{F}|)$*. The new indexer* $\mathbf{I}$ *equals* $\mathbf{I}_{\mathcal{R}}$*; the new prover* $\mathbf{P}$ *runs in time* $\mathrm{time}(\mathbf{P}^{\mathrm{LDT}}) + \mathrm{time}(\mathbf{P}^{\mathcal{R}})$*; and the new verifier* $\mathbf{V}$ *runs in time* $\mathrm{time}(\mathbf{V}^{\mathrm{LDT}}) + \mathsf{q}_{\pi}^{\mathrm{LDT}} \cdot t_{\mathsf{q}}^{\mathcal{R}}$*. (Parameters with superscript "*$\mathcal{R}$*" and "*LDT*" are parameters for* $(\mathbf{I}_{\mathcal{R}}, \mathbf{P}_{\mathcal{R}}, \mathbf{V}_{\mathcal{R}})$ *and* $(\mathbf{P}_{\mathrm{LDT}}, \mathbf{V}_{\mathrm{LDT}})$ *respectively.)*

If $(\mathbf{I}_{\mathcal{R}}, \mathbf{P}_{\mathcal{R}}, \mathbf{V}_{\mathcal{R}})$ *is zero-knowledge, then* $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ *is also zero-knowledge (with the same query bound).*

If $(\mathbf{I}_{\mathcal{R}}, \mathbf{P}_{\mathcal{R}}, \mathbf{V}_{\mathcal{R}})$ *has* **round-by-round knowledge error** $\kappa_{\mathrm{rbr}}^{\mathcal{R}}$ *then* $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ *has round-by-round knowledge error* $\max(\kappa_{\mathrm{rbr}}^{\mathcal{R}}, \epsilon_{\mathrm{rbr}}^{\mathrm{LDT}}, |L|/|\mathbb{F}|)$*.*

Proof. The proof is essentially identical to [BCRSVW19, Theorem 8.1]; note, however, that we do not need to low-degree test the encoded index since it is honestly generated.

To show round-by-round soundness, we define a state function State using the state functions $\mathsf{State}^{\mathcal{R}}$ and $\mathsf{State}^{\mathrm{LDT}}$ of the holographic IOP and low-degree test, respectively. Since the protocols are sequentially composed, we can split the transcript into three parts: $\mathsf{tr}^{\mathcal{R}}$, the first $\mathsf{k}^{\mathcal{R}}$ rounds; $\vec{z}$, the verifier message in round $\mathsf{k}^{\mathcal{R}} + 1$ (to make a full round we precede this with a "dummy" prover message); and $\mathsf{tr}^{\mathrm{LDT}}$, the last $\mathsf{k}^{\mathrm{LDT}}$ rounds. The state function is described by the following algorithm:

$\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}^{\mathcal{R}}, \vec{z}, \mathsf{tr}^{\mathrm{LDT}})$:

1. Let $(\Pi_1, m_1, \ldots, \Pi_j, m_j) := \mathsf{tr}^{\mathcal{R}}$ (for some $j \leq \mathsf{k}^{\mathcal{R}}$). If Π_i is δ^{LDT}-close to $\mathrm{RS}[L, \vec{d}_{\mathbf{P},i}]$ for all $i \in [j]$, output $\mathsf{State}^{\mathcal{R}}(\Pi'_1, m_1, \ldots, \Pi'_j, m_j)$ where $\Pi'_i \in \mathrm{RS}[L, \vec{d}_{\mathbf{P},i}]$ is the closest codeword to Π_i.
2. If $\vec{z}$ is empty then output reject; if $\vec{z}^T\Pi$ is δ^{LDT}-close to $\mathrm{RS}[L, d_{\mathsf{c}}]$ then output accept, where Π is the "stacked" proof matrix (see [BCRSVW19, Protocol 8.2]).
3. Otherwise, output $\mathsf{State}^{\mathrm{LDT}}(\mathbb{i}, \mathbb{x}, \mathsf{tr}^{\mathrm{LDT}})$.

Clearly $\mathsf{State}(\mathbb{i}, \mathbb{x}, \emptyset) = \mathsf{State}^{\mathcal{R}}(\mathbb{i}, \mathbb{x}, \emptyset) = \mathsf{reject}$. For any partial transcript tr, if tr ends during the first stage of the protocol then $\mathsf{rbr}(\mathsf{tr}) \leq \epsilon^{\mathcal{R}}_{\mathrm{rbr}}$. If tr ends with round $\mathsf{k}^{\mathcal{R}}$ and $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$ then the probability that $\vec{z}^T\Pi$ is δ^{LDT}-close to $\mathrm{RS}[L, d_{\mathsf{c}}]$ is bounded by $|L|/|\mathbb{F}|$; hence $\mathsf{rbr}(\mathsf{tr}) \leq |L|/|\mathbb{F}|$. Finally, if tr ends after round $\mathsf{k}^{\mathcal{R}} + 1$, if $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$ then $\vec{z}^T\Pi$ is δ^{LDT}-far from $\mathrm{RS}[L, d_{\mathsf{c}}]$, and so by the RBR soundness guarantee of the low-degree test $\mathsf{rbr}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) \leq \epsilon^{\mathrm{LDT}}_{\mathrm{rbr}}$.

The same state function witnesses round-by-round knowledge soundness. Suppose that for some transcript tr with $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$, $\mathsf{rbr}(\mathsf{tr}) > \max(\kappa^{\mathcal{R}}_{\mathrm{rbr}}, \epsilon^{\mathrm{LDT}}_{\mathrm{rbr}}, |L|/|\mathbb{F}|)$. Since $\mathsf{rbr}(\mathsf{tr}) > \max(\epsilon^{\mathrm{LDT}}_{\mathrm{rbr}}, |L|/|\mathbb{F}|)$, it must be that Π_i is δ^{LDT}-close to $\mathrm{RS}[L, \vec{d}_{\mathbf{P},i}]$ for all i; hence $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}^{\mathcal{R}}_c) = \mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{accept}$. We apply the knowledge extractor of the RS-hIOP to $\tilde{\mathbf{P}}_c$, which runs $\tilde{\mathbf{P}}$ and corrects its output words. The knowledge soundness guarantee for the RS-hIOP ensures that this extractor succeeds. □

Proof of Theorem 8.1. The two main ingredients in the proof are the RS-hIOP of Theorem 7.1 and the FRI low-degree test [BBHR18]. These are combined using Theorem 8.2 to build the described IOP. The indexer in our construction will choose L to be a coset of a smooth subgroup of $\mathbb{F}$ with, say, $8|K| \leq |L| \leq 16|K|$, and $(H \cup K) \cap L = \emptyset$. □

9 Definition of preprocessing non-interactive arguments in the ROM

We denote by $\mathcal{U}(\lambda)$ the set of all functions that map $\{0,1\}^*$ to $\{0,1\}^\lambda$. A *random oracle* with security parameter λ is a function $\rho\colon \{0,1\}^* \to \{0,1\}^\lambda$ sampled uniformly at random from $\mathcal{U}(\lambda)$.

A tuple of algorithms $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ is a *preprocessing non-interactive argument* in the random oracle model (ROM) for an indexed relation $\mathcal{R}$ if the following properties hold.

- **Completeness.** For every adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} & \rho \leftarrow \mathcal{U}(\lambda) \\ \vee & (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^\rho \\ \mathcal{V}^\rho(\mathsf{ivk}, \mathbb{x}, \pi) = 1 & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^\rho(\mathbb{i}) \\ & \pi \leftarrow \mathcal{P}^\rho(\mathsf{ipk}, \mathbb{x}, \mathbb{w}) \end{array}\right] = 1 \ .$$

- **Soundness.** For every t-query adversary $\tilde{\mathcal{P}}$,

$$\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}) & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge & (\mathbb{i}, \mathbb{x}, \pi) \leftarrow \tilde{\mathcal{P}}^\rho \\ \mathcal{V}^\rho(\mathsf{ivk}, \mathbb{x}, \pi) = 1 & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^\rho(\mathbb{i}) \end{array}\right] \leq \epsilon(t, \lambda) \ .$$

The above formulation of completeness allows $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ to depend on the random oracle ρ, and the above formulation of soundness allows $(\mathbb{i}, \mathbb{x})$ to depend on the random oracle ρ.

All constructions in this paper achieve the stronger property of *knowledge soundness*, and optionally also the property of (statistical) *zero knowledge*. We define both of these properties below.

Knowledge soundness. We say that $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ has knowledge error κ if there exists an efficient extractor $\mathcal{E}$ such that for every t-query adversary $\tilde{\mathcal{P}}$

$$\Pr\left[\begin{array}{c|r} \exists\, j \text{ s.t. } (\mathbb{i}_j, \mathbb{x}_j, \mathbb{w}_j) \notin \mathcal{R} & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge & (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}, \vec{\mathbb{w}}) \leftarrow \mathcal{E}^{\rho, \tilde{\mathcal{P}}}(1^t, 1^\lambda) \\ \mathcal{V}^\rho(\mathsf{ivk}_j, \mathbb{x}_j, \pi_j) = 1 & \forall\, j\ ,\ (\mathsf{ipk}_j, \mathsf{ivk}_j) \leftarrow \mathcal{I}^\rho(\mathbb{i}_j) \end{array}\right] \leq \kappa(\lambda)$$

and, moreover, the following distributions are statistically close (as a function of λ)

$$\left\{ (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) \ \middle|\ \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) \leftarrow \tilde{\mathcal{P}}^\rho \end{array} \right\} \quad \text{and} \quad \left\{ (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) \ \middle|\ \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}, \vec{\mathbb{w}}) \leftarrow \mathcal{E}^{\rho, \tilde{\mathcal{P}}}(1^t, 1^\lambda) \end{array} \right\} \ .$$

The above definition is polynomially related to the standard definition of knowledge soundness, which does not consider a vector of outputs or an auxiliary output by the prover.

Zero knowledge. We say that $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for every $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ the distributions below are statistically close (as a function of λ):

$$\left\{ (\rho, \pi) \ \middle|\ \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^\rho(\mathbb{i}) \\ \pi \leftarrow \mathcal{P}^\rho(\mathsf{ipk}, \mathbb{x}, \mathbb{w}) \end{array} \right\} \quad \text{and} \quad \left\{ (\rho[\mu], \pi) \ \middle|\ \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\mu, \pi) \leftarrow \mathcal{S}^\rho(\mathbb{i}, \mathbb{x}) \end{array} \right\} \ .$$

Above, $\rho[\mu]$ is the function that, on input x, equals $\mu(x)$ if μ is defined on x, or $\rho(x)$ otherwise. This definition uses explicitly-programmable random oracles [BR93]. (Non-interactive zero knowledge with non-programmable random oracles is impossible for non-trivial languages [Pas03; BCS16].)

Post-quantum security. The above definitions consider security against *classical* adversaries that make a bounded number of queries to the oracle (and are otherwise computationally unbounded). We also consider security against *quantum* adversaries, whose queries to the oracle can be in superposition. This setting is known as the quantum random oracle model (QROM) [BDFLSZ11], and is the established model to study post-quantum security for constructions that use random oracles. The soundness definition and knowledge soundness definition for post-quantum security are identical to the ones above, except that $\tilde{\mathcal{P}}^{\rho}$ is now taken to mean that $\tilde{\mathcal{P}}$ has superposition query access to ρ; the zero knowledge definition remains unchanged because indistinguishability holds against unbounded adversaries that see the whole oracle.

We do not know if, in the quantum setting, knowledge soundness with auxiliary output is polynomially related to knowledge soundness without auxiliary output.

10 From holographic IOPs to preprocessing arguments

We describe how to transform any public-coin holographic IOP (Section 4) into a corresponding preprocessing non-interactive argument in the ROM (Section 9). After that we explain how the same transformation achieves post-quantum security in the QROM (see Theorem 10.4 towards the end of the section).

Theorem 10.1. *There exists a polynomial-time transformation* T *such that if* $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ *is a public-coin holographic IOP for an indexed relation* $\mathcal{R}$ *then* $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V}) := \mathrm{T}(\mathsf{HOL})$ *is a preprocessing non-interactive argument in the ROM for* $\mathcal{R}$*. In more detail, the transformation* T *has the following properties:*

- *if* HOL *has state-restoration soundness error* $\epsilon_{\mathrm{sr}}(t)$ *then* ARG *has soundness error* $\epsilon_{\mathrm{sr}}(t) + O(t^2 \cdot 2^{-\lambda})$*;*
- *if* HOL *has oracle length* l *and query complexity* q *then* ARG *has argument size* $O(\lambda \cdot \mathsf{q} \cdot \log \mathsf{l})$*;*
- *the time complexities of the argument indexer, prover, and verifier are as follows*

$$\begin{aligned} \mathrm{time}(\mathcal{I}) &= \mathrm{time}(\mathbf{I}) + O(\lambda \mathsf{l}) \ , \\ \mathrm{time}(\mathcal{P}) &= \mathrm{time}(\mathbf{P}) + O(\lambda \mathsf{l}) \ , \\ \mathrm{time}(\mathcal{V}) &= \mathrm{time}(\mathbf{V}) + O(\lambda \cdot \mathsf{q} \cdot \log \mathsf{l}) \ . \end{aligned}$$

Moreover the transformation T *preserves zero knowledge and knowledge soundness in the following sense: if* HOL *is honest-verifier zero knowledge then* ARG *is statistical zero knowledge; and if* HOL *has state-restoration knowledge error* $\kappa_{\mathrm{sr}}(t)$ *then* ARG *has knowledge error* $\kappa_{\mathrm{sr}}(t) + O(t^2 \cdot 2^{-\lambda})$.

The transformation T has two parts. First, we apply the BCS transformation [BCS16] to the holographic IOP to obtain a "holographic" non-interactive argument, namely, a non-interactive argument where the (deterministic) argument verifier is fast when given oracle access to the encoded index. Next, we transform this into a preprocessing non-interactive argument by having the argument indexer output a Merkle commitment to the encoded index, and having the argument prover additionally output Merkle openings to the positions of the encoded index queried by the IOP verifier.

We now describe the transformation T in more detail: in Construction 10.2 we recall the transformation $\mathrm{T}_{\mathsf{BCS}}$ of [BCS16], adapting its presentation to holographic IOPs (but the construction is identical otherwise); then in Construction 10.3 we describe the transformation T, using $\mathrm{T}_{\mathsf{BCS}}$ as a subroutine.

Construction 10.2 ($\mathrm{T}_{\mathsf{BCS}}$)**.** The transformation $\mathrm{T}_{\mathsf{BCS}}$ takes as input a holographic IOP $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ and outputs the (standard) non-interactive argument $\mathsf{ARG} = (\mathcal{P}_{\mathsf{BCS}}, \mathcal{V}_{\mathsf{BCS}})$ defined below.

```
$\mathcal{P}^{\rho}_{\mathsf{BCS}}(\mathbb{i}, \mathbb{x}, \mathbb{w})$:
1. Set $\sigma_0 := \mathsf{ivk} \| \mathbb{x}$, where $(\mathsf{ivk}, \mathsf{ipk}) \leftarrow \mathcal{I}(\mathbb{i})$.
2. For $i = 1, \dots, \mathsf{k}$:
   (a) Compute randomness $\rho_i := \rho(\sigma_{i-1})$ for the $i$-th round.
   (b) Provide $\rho_i$ to the IOP prover $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ to obtain a proof oracle $\Pi_i$.
   (c) Use $\rho$ to compute a Merkle tree on $\Pi_i$, and in particular to obtain a Merkle root $\mathsf{rt}_i$.
   (d) Set $\sigma_i := \rho(\sigma_{i-1} \| \mathsf{rt}_i)$.
3. Compute randomness $\rho_{\mathsf{k}+1} := \rho(\sigma_{\mathsf{k}})$ for the query phase.
4. Run the IOP verifier $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x}; \rho_1, \dots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1})$, answering queries via the proof oracles
   $(\Pi_1, \dots, \Pi_{\mathsf{k}})$, so to deduce the set of queries $Q$ that are asked on randomness $(\rho_1, \dots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1})$.
5. Output the proof string $\pi$ that contains all the Merkle roots $(\mathsf{rt}_1, \cdots, \mathsf{rt}_{\mathsf{k}})$ and, for each query
   in $Q$, an answer supported by an authentication path (against the appropriate root).
```

$\mathcal{V}^{\rho}_{\mathsf{BCS}}(\mathbb{i}, \mathbb{w}, \pi) \equiv \mathcal{V}^{\rho,\mathbf{I}(\mathbb{i})}_{\mathsf{BCS}}(\mathbb{x}, \pi)$:

1. Set $\sigma_0 := \mathsf{ivk} \| \mathbb{x}$ and use the i-th root rt_i in π to set $\sigma_i := \rho(\sigma_{i-1} \| \mathsf{rt}_i)$ for $i = 1, \ldots, \mathsf{k}$.
2. Compute each randomness: $\rho_1 := \rho(\sigma_0), \ldots, \rho_{\mathsf{k}} := \rho(\sigma_{\mathsf{k}-1}), \rho_{\mathsf{k}+1} := \rho(\sigma_{\mathsf{k}})$.
3. Run the IOP verifier $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1})$. Whenever $\mathbf{V}$ queries a proof oracle Π_i, validate the authentication path for this query in π, and answer the query with the corresponding value in π. (If π contains no entry for a query, reject.) Accept if and only if $\mathbf{V}$ accepts.

We write $\mathcal{V}_{\mathsf{BCS}}$ as an algorithm with oracle access to $\mathbf{I}(\mathbb{i})$ to emphasize that $\mathrm{T}_{\mathsf{BCS}}$ is black-box with respect to $\mathbf{V}$: the queries $\mathcal{V}_{\mathsf{BCS}}$ makes to $\mathbf{I}(\mathbb{i})$ are exactly the queries $\mathbf{V}$ makes to $\mathbf{I}(\mathbb{i})$ (on appropriate randomness).

Construction 10.3 (T)**.** The transformation T takes as input a public-coin holographic IOP $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ and outputs the preprocessing non-interactive argument $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ defined below. We let $(\mathcal{P}_{\mathsf{BCS}}, \mathcal{V}_{\mathsf{BCS}})$ be the BCS prover and verifier output by $\mathrm{T}_{\mathsf{BCS}}(\mathbf{I}, \mathbf{P}, \mathbf{V})$, and view $\mathcal{V}_{\mathsf{BCS}}$ as having oracle access to $\mathbf{I}(\mathbb{i})$.

- *Indexer.* On input $\mathbb{i}$, $\mathcal{I}^{\rho}$ computes the encoded index $\mathbf{I}(\mathbb{i})$, computes a Merkle commitment rt to $\mathbf{I}(\mathbb{i})$ using ρ, and outputs the key pair $(\mathsf{ipk}, \mathsf{ivk}) := ((\mathbb{i}, \mathbf{I}(\mathbb{i})), \mathsf{rt})$.
- *Prover.* On input $(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$, $\mathcal{P}^{\rho}$ parses the proving key ipk as $(\mathbb{i}, \mathbf{I}(\mathbb{i}))$, computes the output of the BCS prover $\pi_{\mathsf{BCS}} := \mathcal{P}^{\rho}_{\mathsf{BCS}}(\mathbb{i}, \mathbb{x}, \mathbb{w})$, simulates the BCS verifier $\mathcal{V}^{\rho,\mathbf{I}(\mathbb{i})}_{\mathsf{BCS}}(\mathbb{x}, \pi_{\mathsf{BCS}})$ letting ap_i be the authentication path for its i-th query to $\mathbf{I}(\mathbb{i})$, and outputs the proof string $\pi := (\pi_{\mathsf{BCS}}, (\mathsf{ap}_1, \ldots, \mathsf{ap}_k))$.
- *Verifier.* On input $(\mathsf{ivk}, \mathbb{x}, \pi)$, $\mathcal{V}^{\rho}$ parses the proof string π as $(\pi_{\mathsf{BCS}}, (\mathsf{ap}_1, \ldots, \mathsf{ap}_k))$, runs the BCS verifier $\mathcal{V}^{\rho;\bullet}_{\mathsf{BCS}}(\mathbb{x}, \pi_{\mathsf{BCS}})$ and answers its i-th query to the second oracle (denoted via the symbol "$\bullet$") using the provided authentication paths. If for any i, ap_i is not a valid authentication path with respect to the Merkle root in ivk and the position requested in the i-th query, then $\mathcal{V}$ rejects. Otherwise, $\mathcal{V}$ accepts if $\mathcal{V}_{\mathsf{BCS}}$ does.

Proof of Theorem 10.1. Completeness of the argument system is straightforward from the protocol description. For soundness, consider an index-instance pair $(\mathbb{i}, \mathbb{x})$ that is not in $\mathcal{L}(\mathcal{R})$ and a t-query malicious prover $\tilde{\mathcal{P}}$. Let E be the event that, over a random oracle $\rho \leftarrow \mathcal{U}(\lambda)$ and letting $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathbb{i})$, for the proof string $\tilde{\pi} = (\pi_{\mathsf{BCS}}, (\mathsf{ap}_1, \ldots, \mathsf{ap}_k))$ output by $\tilde{\mathcal{P}}^{\rho}$ there exists an authentication path ap_i for some query location $j \in |\mathbf{I}(\mathbb{i})|$ that is valid with respect to ivk, ρ but the opened value is not equal to $\mathbf{I}(\mathbb{i})_j$. If E occurs then we can find a collision in ρ via $O(\log |\mathbf{I}(\mathbb{i})|)$ additional queries. Therefore

$$\begin{aligned}
&\Pr_{\rho}[\mathcal{V}^{\rho}(\mathsf{ivk}, \mathbb{x}, \tilde{\pi}) = 1] \\
\leq &\Pr_{\rho}[\mathcal{V}^{\rho}(\mathsf{ivk}, \mathbb{x}, \tilde{\pi}) = 1 \mid \neg E] + \Pr_{\rho}[E] \\
\leq &\Pr_{\rho}[\mathcal{V}^{\rho,\mathbf{I}(\mathbb{i})}_{\mathsf{BCS}}(\mathbb{x}, \tilde{\pi}) = 1] + (t + O(\log \mathsf{l}))^2/2^{\lambda} \quad .
\end{aligned}$$

The soundness guarantee of $\mathrm{T}_{\mathsf{BCS}}$ ensures that $\Pr_{\rho}[\mathcal{V}^{\rho,\mathbf{I}(\mathbb{i})}_{\mathsf{BCS}}(\mathbb{x}, \tilde{\pi}) = 1] \leq \epsilon_{\mathrm{sr}}(t) + O(t^2 \cdot 2^{-\lambda})$, which yields the stated bound (since the query bound t is at least $\log \mathsf{l}$ without loss of generality).

Efficiency. The proof string π output by the argument prover $\mathcal{P}$ has two components: the proof string π_{BCS} output by the BCS prover $\mathcal{P}_{\mathsf{BCS}}$, and authentication paths $(\mathsf{ap}_1, \ldots, \mathsf{ap}_k)$ that answer queries by the IOP verifier $\mathbf{V}$ to the encoded index. Each of these components has size $O(\lambda \cdot \mathsf{q} \cdot \log \mathsf{l})$. We now discuss time complexities. The overhead of the argument indexer $\mathcal{I}$ with respect to the IOP indexer $\mathbf{I}$ is $O(\lambda \cdot \mathsf{l})$, due to the cost of committing to the encoded index output by $\mathbf{I}$. The overhead of the argument prover $\mathcal{P}$ with respect to the IOP prover $\mathbf{P}$ is $O(\lambda \cdot \mathsf{l})$, due to the cost of committing to each oracle output by $\mathbf{P}$ (and the cost to answer queries by the IOP verifier $\mathbf{V}$ to the oracles or the encoded index). The overhead of the argument verifier $\mathcal{V}$ with respect to the IOP verifier $\mathbf{V}$ is $O(\lambda \cdot \mathsf{q} \cdot \log \mathsf{l})$, due to the cost to validate the authentication path associated to each query made by $\mathbf{V}$ (to a proof oracle or the encoded index).

Zero knowledge. The fact that the transformation T preserves zero knowledge follows from the fact that the BCS transformation $\mathrm{T}_{\mathsf{BCS}}$ preserves zero knowledge (if leaves in the Merkle tree are suitably salted), because the simulator is given the index $\mathbb{i}$ as input. See [BCS16] for details on why if HOL is honest-verifier zero knowledge (when viewed as a non-holographic proof system) then $(\mathcal{P}_{\mathsf{BCS}}, \mathcal{V}_{\mathsf{BCS}})$ is statistical zero knowledge.

Knowledge soundness. The fact that the transformation T preserves knowledge soundness follows from the fact that the BCS transformation $\mathrm{T}_{\mathsf{BCS}}$ preserves knowledge soundness and the query lower bound for finding collisions in the random oracle. Namely, the same derivation above for soundness shows that if HOL has state-restoration knowledge error $\kappa_{\mathrm{sr}}(t)$ then ARG has knowledge error $\kappa_{\mathrm{sr}}(t) + O(t^2 \cdot 2^{-\lambda})$. □

Theorem 10.4. *The transformation* T *of Theorem 10.1 also satisfies the following: if* HOL *has round-by-round soundness error* ϵ_{rbr} *then* ARG *has soundness error* $t^2 \cdot \epsilon_{\mathrm{rbr}} + O(t^3 \cdot 2^{-\lambda})$ *in the QROM; if* HOL *has round-by-round knowledge error* κ_{rbr} *then* ARG *has knowledge error* $t^2 \cdot \kappa_{\mathrm{rbr}} + O(t^3 \cdot 2^{-\lambda})$ *in the QROM.*

Proof. The proof is almost identical to the proof of Theorem 10.1, where we now use the result of [CMS19] to bound the probability that the BCS verifier accepts in the QROM and the quantum query lower bound for collisions [AS04] to bound the probability of the event E (which implies that a collision was found). □

11 Recursive composition in the URS model

We describe how to transform any preprocessing SNARK in the URS model into a preprocessing PCD scheme in the URS model. The transformation preserves post-quantum security.

This section is organized as follows. In Section 11.1 we define preprocessing SNARKs in the URS model. In Section 11.2 we define preprocessing PCD schemes in the URS model. In Section 11.3 we state the properties of the transformation from SNARK to PCD. In sec:recursion-efficiency we describe the construction and prove its efficiency properties. In Section 11.5 we prove the security properties.

In this section by "polynomial-size" we mean a (non-uniform) family of polynomial-size circuits.

11.1 Preprocessing non-interactive arguments (of knowledge) in the URS model

Informally, the definition of a preprocessing SNARK in the URS model is similar to the definition of a preprocessing SNARK in the random oracle model (see Section 9) except that the random oracle is replaced by a $\mathsf{poly}(\lambda)$-size uniform random string urs. The formal definition follows.

A tuple of algorithms $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ is a *preprocessing non-interactive argument (of knowledge)* in the uniform random string (URS) model for an indexed relation $\mathcal{R}$ if the following properties hold.

Completeness. For every adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|c} (\mathbb{i},\mathbb{x},\mathbb{w}) \notin \mathcal{R} & \mathsf{urs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ \vee & (\mathbb{i},\mathbb{x},\mathbb{w}) \leftarrow \mathcal{A}(\mathsf{urs}) \\ \mathcal{V}(\mathsf{urs},\mathsf{ivk},\mathbb{x},\pi) = 1 & (\mathsf{ipk},\mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs},\mathbb{i}) \\ & \pi \leftarrow \mathcal{P}(\mathsf{urs},\mathsf{ipk},\mathbb{x},\mathbb{w}) \end{array}\right] = 1 \ .$$

The above formulation of completeness allows $(\mathbb{i},\mathbb{x},\mathbb{w})$ to depend on the reference string urs.

Knowledge soundness. We say that $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ has knowledge error κ if for every polynomial-size adversary $\tilde{\mathcal{P}}$ there exists a polynomial-size extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{c|c} \exists j \text{ s.t. } (\mathbb{i}_j,\mathbb{x}_j,\mathbb{w}_j) \notin \mathcal{R} & \mathsf{urs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ \wedge & (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux},\vec{\mathbb{w}}) \leftarrow \mathcal{E}(\mathsf{urs}) \\ \mathcal{V}(\mathsf{urs},\mathsf{ivk}_j,\mathbb{x}_j,\pi_j) = 1 & \forall j\ ,\ (\mathsf{ipk}_j,\mathsf{ivk}_j) \leftarrow \mathcal{I}(\mathsf{urs},\mathbb{i}_j) \end{array}\right] \le \kappa(\lambda)$$

and, moreover, the following distributions are $\kappa(\lambda)$-close in statistical distance:

$$\left\{ (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux}) \;\middle|\; \begin{array}{c} \mathsf{urs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux}) \leftarrow \tilde{\mathcal{P}}(\mathsf{urs}) \end{array} \right\} \quad \text{and} \quad \left\{ (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux}) \;\middle|\; \begin{array}{c} \mathsf{urs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux},\vec{\mathbb{w}}) \leftarrow \mathcal{E}(\mathsf{urs}) \end{array} \right\} \ .$$

The above definition is polynomially related to the standard definition of knowledge soundness, which does not consider a vector of outputs or an auxiliary output by the prover.

Zero knowledge. We say that $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for every honest adversary $\mathcal{A}$ the distributions below are statistically close (as a function of λ):

$$\left\{ (\mathsf{urs},\pi) \;\middle|\; \begin{array}{c} \mathsf{urs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ (\mathbb{i},\mathbb{x},\mathbb{w}) \leftarrow \mathcal{A}(\mathsf{urs}) \\ (\mathsf{ipk},\mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs},\mathbb{i}) \\ \pi \leftarrow \mathcal{P}(\mathsf{urs},\mathsf{ipk},\mathbb{x},\mathbb{w}) \end{array} \right\} \quad \text{and} \quad \left\{ (\mathsf{urs},\pi) \;\middle|\; \begin{array}{c} (\mathsf{urs},\tau) \leftarrow \mathcal{S}(1^\lambda) \\ (\mathbb{i},\mathbb{x},\mathbb{w}) \leftarrow \mathcal{A}(\mathsf{urs}) \\ \pi \leftarrow \mathcal{S}(\mathbb{i},\mathbb{x},\tau) \end{array} \right\} \ .$$

In this case, an adversary $\mathcal{A}$ is honest if it outputs $(\mathbb{i},\mathbb{x},\mathbb{w}) \in \mathcal{R}$ with probability 1.

Remark 11.1 (post-quantum security)**.** The above definitions consider security against *classical* polynomial-size adversaries. We also consider security against *quantum* polynomial-size adversaries. The definitions for this case are identical, except that $\tilde{\mathcal{P}}$ is a (non-uniform) family of polynomial-size *quantum* circuits. We do not know if, in the quantum setting, knowledge soundness with auxiliary output (which we achieve in the random oracle model) is polynomially related to knowledge soundness without auxiliary output.

11.2 Preprocessing PCD in the URS model

We have informally introduced PCD in in Section 2.5. Formally, a triple of algorithms $\mathsf{PCD} = (\mathbb{I}, \mathbb{P}, \mathbb{V})$ is a *proof-carrying data scheme* (PCD scheme) in the uniform random string (URS) model for a class of compliance predicates F if the properties below hold.

Definition 11.2. *A* **transcript** T *is a directed acyclic graph where each vertex* $u \in V(\mathsf{T})$ *is labeled by a local data* $z_{\mathsf{loc}}^{(u)}$ *and each edge* $e \in E(\mathsf{T})$ *is labeled by a message* $z^{(e)} \neq \bot$*. The* **output** *of a transcript* T*, denoted* $\mathsf{o}(\mathsf{T})$*, is* $z^{(e)}$ *where* $e = (u, v)$ *is the lexicographically-first edge such that* v *is a sink.*

Definition 11.3. *A vertex* $u \in V(\mathsf{T})$ *is* **Φ-compliant** *for* $\Phi \in \mathsf{F}$ *if for all outgoing edges* $e = (u, v) \in E(\mathsf{T})$*:*

- *(base case) if* u *has no incoming edges,* $\Phi(z^{(e)}, z_{\mathsf{loc}}^{(u)}, \bot, \ldots, \bot)$ *accepts;*
- *(recursive case) if* u *has incoming edges* $e_1, \ldots, e_m$*,* $\Phi(z^{(e)}, z_{\mathsf{loc}}^{(u)}, z^{(e_1)}, \ldots, z^{(e_m)})$ *accepts.*

We say that T *is* **Φ-compliant** *if all of its vertices are Φ-compliant.*

Completeness. For every adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|r} \Phi \notin \mathsf{F} & \\ \vee & \mathsf{urs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ \mathsf{T} \text{ is not } \Phi\text{-compliant} & \Phi \leftarrow \mathcal{A}(\mathsf{urs}) \\ \vee & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathsf{urs}, \Phi) \\ \mathbb{V}(\mathsf{urs}, \mathsf{ivk}, \mathsf{o}(\mathsf{T}), \pi) = 1 & (\mathsf{T}, \pi) \leftarrow \mathsf{ProofGen}(\mathsf{urs}, \mathbb{P}, \mathsf{ipk}, \mathcal{A}) \end{array}\right] = 1 \ .$$

Above, $\mathsf{ProofGen}$ is an interactive protocol between $\mathcal{A}$ and $\mathbb{P}$ that proceeds as follows. The protocol maintains a computation graph T with node and edge labels, which is initially empty. In each round of the interaction, $\mathcal{A}$ may choose to add a node or an edge to the computation graph. Nodes correspond to local computations, and labeled edges to passed messages. If $\mathcal{A}$ chooses to add an edge (u, v) with label z, $\mathbb{P}$ must generate a proof that sending z from u to v is compliant with Φ, and label (u, v) with that proof. When $\mathcal{A}$ halts, $\mathsf{ProofGen}$ outputs the labeled graph T along with the proof π corresponding to the output edge.

Knowledge soundness. We say that $\mathsf{PCD} = (\mathbb{I}, \mathbb{P}, \mathbb{V})$ has knowledge error κ if there exists some polynomial e such that for every polynomial-size adversary $\tilde{\mathbb{P}}$ there exists an extractor $\mathbb{E}$ of size at most $e(|\tilde{\mathbb{P}}|)$ such that

$$\Pr\left[\begin{array}{c|r} \Phi \in \mathsf{F} & \\ \wedge & \mathsf{urs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ \mathsf{T} \text{ is not } \Phi\text{-compliant or } \mathsf{o}(\mathsf{T}) \neq \mathsf{o} & (\Phi, \mathsf{o}, \pi, \mathsf{T}) \leftarrow \mathbb{E}(\mathsf{urs}) \\ \wedge & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathsf{urs}, \Phi) \\ \mathbb{V}(\mathsf{urs}, \mathsf{ivk}, \mathsf{o}, \pi) = 1 & \end{array}\right] \leq \kappa(\lambda) \ .$$

and, moreover, the following distributions are $\kappa(\lambda)$-close in statistical distance:

$$\left\{ (\Phi, \mathsf{o}, \pi) \;\middle|\; \begin{array}{c} \mathsf{urs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ (\Phi, \mathsf{o}, \pi) \leftarrow \tilde{\mathbb{P}}(\mathsf{urs}) \end{array} \right\} \quad \text{and} \quad \left\{ (\Phi, \mathsf{o}, \pi) \;\middle|\; \begin{array}{c} \mathsf{urs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ (\Phi, \mathsf{o}, \pi, \mathsf{T}) \leftarrow \mathbb{E}(\mathsf{urs}) \end{array} \right\} \ .$$

Efficiency. The indexer $\mathbb{I}$ runs in time $\mathsf{poly}(\lambda, |\Phi|)$. The prover $\mathbb{P}$ runs in time $\mathsf{poly}(\lambda, |\Phi|)$. The verifier $\mathbb{V}$ runs in time $\mathsf{poly}(\lambda, |\mathsf{o}|, \log |\Phi|)$. A proof π has size $\mathsf{poly}(\lambda, \log |\Phi|)$.

Zero knowledge. We say that $\mathsf{PCD} = (\mathbb{I}, \mathbb{P}, \mathbb{V})$ has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathbb{S}$ such that for every honest adversary $\mathcal{A}$ the distributions below are statistically close (as a function of λ):

$$\left\{ (\mathsf{urs}, \pi) \;\middle|\; \begin{array}{r} \mathsf{urs} \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)} \\ (\Phi, \mathsf{o}) \leftarrow \mathcal{A}(\mathsf{urs}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathsf{urs}, \Phi) \\ (\mathsf{T}, \pi) \leftarrow \mathsf{ProofGen}(\mathsf{urs}, \mathbb{P}, \mathsf{ipk}, \mathcal{A}) \end{array} \right\} \quad \text{and} \quad \left\{ (\mathsf{urs}, \pi) \;\middle|\; \begin{array}{r} (\mathsf{urs}, \tau) \leftarrow \mathbb{S} \\ (\Phi, \mathsf{o}) \leftarrow \mathcal{A}(\mathsf{urs}) \\ \pi \leftarrow \mathbb{S}(\Phi, \mathsf{o}, \tau) \end{array} \right\} .$$

In this case, $\mathcal{A}$ is honest if it outputs, with probability 1, (Φ, o) such that the resulting transcript T is Φ-compliant and $\mathsf{o}(\mathsf{T}) = \mathsf{o}$.

11.3 Theorem statement

The key parameter that determines the efficiency of the preprocessing PCD scheme is the size of the preprocessing SNARK verifier as a circuit (or constraint system), as captured by the following definition.

Definition 11.4. *Let $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ be a preprocessing non-interactive argument in the URS model. We denote by $\mathcal{V}^{(\lambda,N,k)}$ the circuit (or constraint system) corresponding to the computation of the SNARK verifier $\mathcal{V}$, for security parameter λ, when checking indices of size at most N and instances of size at most k. Hence, for every $\mathsf{urs} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ and index-instance pair $(\mathbb{i}, \mathbb{x})$ with $|\mathbb{i}| \leq N$ and $|\mathbb{x}| \leq k$, index key pair $(\mathsf{ipk}, \mathsf{ivk}) \in \mathcal{I}(\mathsf{urs}, \mathbb{i})$, and candidate proof π, we have $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, \mathbb{x}, \pi) = \mathcal{V}(\mathsf{urs}, \mathsf{ivk}, \mathbb{x}, \pi)$. We denote by $\mathsf{v}(\lambda, N, k)$ the size of the circuit $\mathcal{V}^{(\lambda,N,k)}$, and by $|\mathsf{ivk}(\lambda, N)|$ the size of the index verification key ivk.*

The *depth* of a compliance predicate $\Phi \colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$, denoted $d(\Phi)$, is the maximum depth of any Φ-compliant transcript T. We prove the following theorem, which constructs a PCD system for constant-depth compliance predicates from any sufficiently efficient preprocessing SNARK.

Theorem 11.5. *There exists a polynomial-time transformation T such that if $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ is a preprocessing SNARK for $\mathcal{R}_{\mathrm{R1CS}}$ in the URS model then $\mathsf{PCD} = (\mathbb{I}, \mathbb{P}, \mathbb{V}) := \mathrm{T}(\mathsf{ARG})$ is a preprocessing PCD scheme in the URS model for constant-depth compliance predicates, provided*

$$\exists\, \epsilon \in (0,1) \text{ and a polynomial } \alpha \text{ s.t. } \mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell) = O(N^{1-\epsilon} \cdot \alpha(\lambda, \ell)) \ .$$

Moreover, if the size of the predicate $\Phi \colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$ is $f = \omega((m \cdot \alpha(\lambda, \ell))^{1/\epsilon})$ then the PCD indexer, PCD prover, and PCD verifier run in time that equal those of the SNARK indexer, SNARK prover, and SNARK verifier on R1CS indices $\mathbb{i}$ of size $f + o(f)$ (and R1CS instances $\mathbb{x}$ of size $O(\lambda) + \ell$).

If ARG is zero knowledge, then PCD is zero knowledge.

If ARG is secure against quantum adversaries, then PCD is secure against quantum adversaries.

Remark 11.6. Our preprocessing zkSNARK for R1CS, FRACTAL, achieves the following verifier size:

$$\mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell) = O(\lambda\ell + \lambda^2 \log^2(N)) \ ,$$

assuming a choice of cryptographic hash function that can be expressed via a constraint system of size $O(\lambda)$. This means that we may take any $\epsilon \in (0,1)$ and $\alpha(\lambda, \ell) := \lambda(\lambda + \ell)$. In particular, if the size of a compliance predicate Φ grows as $(m\lambda(\lambda + \ell))^{1+\delta}$ for any $\delta > 0$, then the time bounds in Theorem 11.5 hold for us.

11.4 Construction and its efficiency

We describe how to construct the preprocessing PCD scheme, and then prove the efficiency properties stated in Theorem 11.5. We defer proving the security properties to Section 11.5.

Construction 11.7 (from SNARK to PCD)**.** Let $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ be a preprocessing SNARK for R1CS. We describe how to construct a preprocessing PCD scheme $\mathsf{PCD} = (\mathbb{I}, \mathbb{P}, \mathbb{V})$.

Given a compliance predicate $\Phi \colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$, the circuit that realizes the recursion is as follows.

```
$R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}\big((\mathsf{ivk}, z_{\mathsf{out}}), (z_{\mathsf{loc}}, (z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}})_{i\in[m]})\big)$:
1. Check that the compliance predicate $\Phi(z_{\mathsf{out}}, z_{\mathsf{loc}}, z^{(1)}_{\mathsf{in}}, \dots, z^{(m)}_{\mathsf{in}})$ accepts.
2. If there exists $i$ such that $(z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}}) \neq \bot$:
       check that, for every $i \in [m]$, the SNARK verifier $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z^{(i)}_{\mathsf{in}}), \pi^{(i)}_{\mathsf{in}})$ accepts.
3. If the above checks hold, output 0; otherwise, output 1.
```

Next we describe the indexer $\mathbb{I}$, prover $\mathbb{P}$, and verifier $\mathbb{V}$ of the PCD scheme.

- $\mathbb{I}(\mathsf{urs}, \Phi)$:
  1. Compute $N := N(\lambda, |\Phi|, m, \ell)$, where N is as defined in Lemma 11.8 below.
  2. Construct the circuit $R := R^{(\lambda,N,|\mathsf{ivk}(\lambda,N)|+\ell)}_{\mathcal{V},\Phi,\mathsf{urs}}$.
  3. Compute the index key pair $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs}, R)$.
- $\mathbb{P}(\mathsf{urs}, \mathsf{ipk}, z_{\mathsf{out}}, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, \vec{\pi}_{\mathsf{in}})$: output the proof $\pi_{\mathsf{out}} \leftarrow \mathcal{P}\big(\mathsf{urs}, \mathsf{ipk}, (\mathsf{ivk}, z_{\mathsf{out}}), (z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, \vec{\pi}_{\mathsf{in}})\big)$.
- $\mathbb{V}(\mathsf{urs}, \mathsf{ivk}, z_{\mathsf{out}}, \pi_{\mathsf{out}})$: accept if $\mathcal{V}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z_{\mathsf{out}}), \pi_{\mathsf{out}})$ accepts.

Proof of Theorem 11.5 (efficiency). Denote by f the size of Φ as an R1CS instance. In Construction 11.7, the explicit input consists of the index verification key ivk, whose size depends on N and λ, and a message z whose size is ℓ (independent of N). The security parameter λ is also independent of N. The circuit on which we wish to invoke $\mathcal{V}$ is of size

$$S(\lambda, f, m, \ell, N) = S_0(f, m, \ell) + m \cdot \mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell) \quad \text{for some} \quad S_0(f, m, \ell) = f + O(m\ell) \ .$$

We want to find a function N such that $S(\lambda, f, m, \ell, N(\lambda, f, m, \ell)) \leq N(\lambda, f, m, \ell)$ and N is not too large.

Lemma 11.8. *Suppose that for every security parameter $\lambda \in \mathbb{N}$ and message size $\ell \in \mathbb{N}$ the ratio of verifier circuit size to index size $\mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell)/N$ is monotone decreasing in N. Then there exists a size function $N(\lambda, f, m, \ell)$ such that*

$$\forall\, \lambda, f, m, \ell \in \mathbb{N} \quad S(\lambda, f, m, \ell, N(\lambda, f, m, \ell)) \leq N(\lambda, f, m, \ell) \ .$$

Moreover if for some $\epsilon > 0$ and some increasing function α it holds that, for all N, λ, ℓ sufficiently large,

$$\mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell) \leq N^{1-\epsilon} \alpha(\lambda, \ell)$$

then, for all λ, ℓ sufficiently large,

$$N(\lambda, f, m, \ell) \leq O(f) + (2m \cdot \alpha(\lambda, \ell))^{1/\epsilon} \ .$$

Proof. Let $N_0 := N_0(\lambda, m, \ell)$ be the smallest integer such that $\mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell)/N < 1/(2m)$; this exists because of the monotone decreasing condition. Let $N(\lambda, f, m, \ell) := \max(N_0(\lambda, m, \ell), 2S_0(f, m, \ell))$. Then for $N := N(\lambda, f, m, \ell)$ it holds that

$$S(\lambda, f, m, \ell, N) = S_0(f, m, \ell) + mN \cdot \mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell)/N < N/2 + N/2 = N \ .$$

Clearly $S_0(f, m, \ell) = O(f)$. Now suppose that $\mathsf{v}(\lambda, N, |\mathsf{ivk}(N)| + \ell) \leq (N^{1-\epsilon}\alpha(\lambda, \ell))$ for all sufficiently large N, λ, ℓ. Let $N'(\lambda, m, \ell) := (2m \cdot \alpha(\lambda, \ell))^{1/\epsilon}$. Then for all m and sufficiently large λ, ℓ, for $N' := N'(\lambda, m, \ell)$,

$$\mathsf{v}(\lambda, N', |\mathsf{ivk}(\lambda, N')| + \ell)/N' < \alpha(\lambda, \ell) \cdot (2m \cdot \alpha(\lambda, \ell))^{-1} = 1/(2m) \ .$$

Hence $N_0 \leq N' = (2m \cdot \alpha(\lambda, \ell))^{1/\epsilon}$. □

The size of the circuit $R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}$ for $N := N(\lambda, f, m, \ell)$ and $k := |\mathsf{ivk}(\lambda, N)| + \ell$ is at most

$$\begin{aligned} S(\lambda, f, m, \ell, N) &= f + O(m\ell) + m \cdot \mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell) \\ &= f + O(N^{1-\epsilon} m\alpha(\lambda, \ell)) \\ &= f + O(f^{1-\epsilon} m \cdot \alpha(\lambda, \ell) + (m \cdot \alpha(\lambda, \ell))^{1/\epsilon}) \ . \end{aligned}$$

In particular if $f = \omega((m \cdot \alpha(\lambda, \ell))^{1/\epsilon})$ then this is $f + o(f)$, and so the stated efficiency bounds hold. □

11.5 Security reduction

We establish the security properties in Theorem 11.5. We discuss knowledge soundness in Section 11.5.1, post-quantum security in Section 11.5.2, and zero knowledge in Section 11.5.3.

11.5.1 Knowledge soundness

In the following, since the extracted transcript T will be a tree, we find it convenient to associate the label $(z^{(u,v)}, \pi^{(u,v)})$ of the unique outgoing edge of a node u with the node u itself, so we refer to this as $(z^{(u)}, \pi^{(u)})$. It is straightforward to transform such a transcript into one that satisfies Definition 11.2.

Given a malicious prover $\tilde{\mathbb{P}}$, we will define an extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ that satisfies knowledge soundness. In the process we construct a sequence of extractors $\mathbb{E}_1, \ldots, \mathbb{E}_d$ for $d := d(\Phi)$ (the depth of Φ); $\mathbb{E}_j$ outputs a tree of depth $j + 1$. Let $\mathbb{E}_0(\mathsf{urs})$ run $(\Phi, \mathsf{o}, \pi) \leftarrow \tilde{\mathbb{P}}(\mathsf{urs})$ and output (Φ, T_0), where T_0 is a single node labeled with (o, π). Let $l_{\mathsf{T}}(j)$ denote the vertices of T at depth j; $l_{\mathsf{T}}(0) := \emptyset$ and $l_{\mathsf{T}}(1)$ is the singleton containing the root.

Now we define the extractor $\mathbb{E}_j$ inductively for each $j \in \{1, \ldots, d\}$. Suppose we have already constructed $\mathbb{E}_{j-1}$. We construct a SNARK prover $\tilde{\mathcal{P}}_j$ as follows:

$\tilde{\mathcal{P}}_j(\mathsf{urs})$:

1. Run $(\Phi, \mathsf{T}_{j-1}) \leftarrow \mathbb{E}_{j-1}(\mathsf{urs})$; for each vertex $v \in l_{\mathsf{T}_{j-1}}(j)$, denote its label by $(z^{(v)}, \pi^{(v)})$.
2. Run the indexer $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs}, R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}})$.
3. Output

$$(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) := \Big(\vec{R}, (\mathsf{ivk}, z^{(v)})_{v \in l_{\mathsf{T}_{j-1}}(j)}, (\pi^{(v)})_{v \in l_{\mathsf{T}_{j-1}}(j)}, (\Phi, \mathsf{T}_{j-1})\Big)$$

where $\vec{R}$ is the vector $(R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}, \ldots, R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}})$ of the appropriate length.

Next let $\mathcal{E}_{\tilde{\mathcal{P}}_j}$ be the extractor that corresponds to $\tilde{\mathcal{P}}_j$, via the knowledge soundness of the non-interactive argument ARG. Finally the extractor $\mathbb{E}_j$ is defined as follows:

$\mathbb{E}_j(\mathsf{urs})$:

1. Run the extractor $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}, \vec{\mathbb{w}}) \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}_j}(\mathsf{urs})$, and parse the auxiliary output aux as (Φ, T').
2. If T' is not a transcript of depth j, abort.
3. Output (Φ, T_j) where T_j is the transcript constructed from T' by doing the following for each vertex $v \in l_{\mathsf{T}'}(j)$:
   - obtain the local data $z^{(v)}_{\mathsf{loc}}$ and input messages $\left(z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}}\right)_{i\in[m]}$ from $\mathbb{w}^{(v)}$;
   - append $z^{(v)}_{\mathsf{loc}}$ to the label of v, and if there exists any $z^{(i)}_{\mathsf{in}}$ with $z^{(i)}_{\mathsf{in}} \neq \bot$, attach m children to v where the i-th child is labeled with $(z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}})$.

The extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ runs $(\Phi, \mathsf{T}_d) \leftarrow \mathbb{E}_d(\mathsf{urs})$ and outputs $(\Phi, \mathsf{o}, \pi, \mathsf{T}_d)$, where (o, π) labels the root node.

We now show that $\mathbb{E}_{\tilde{\mathbb{P}}}$ has polynomial size and that it outputs a transcript that is Φ-compliant.

Size of the extractor. $\tilde{\mathcal{P}}_j$ is a circuit of size $|\mathbb{E}_{j-1}| + |\mathcal{I}| + O(2^j)$, so $\mathcal{E}_{\tilde{\mathcal{P}}_j}$ is a circuit of size $e(|\mathbb{E}_{j-1}| + |\mathcal{I}| + O(2^j))$ Then $|\mathbb{E}_j| \leq e(|\mathbb{E}_{j-1}| + |\mathcal{I}| + c \cdot 2^j)$ for some $c \in \mathbb{N}$.

A solution to this recurrence (for $e(n) \geq n$) is $|\mathbb{E}_j| \leq e^{(j)}(|\tilde{\mathbb{P}}| + j \cdot |\mathcal{I}| + 2c \cdot 2^j)$, where $e^{(j)}$ is the function e iterated j times. Hence in particular if $d(\Phi)$ is a constant, $\mathbb{E}_{\tilde{\mathbb{P}}}$ is a circuit of polynomial size.

Correctness of the extractor. We show by induction that, for all $j \in \{0, \ldots, d\}$, the transcript T_j output by $\mathbb{E}_j$ is Φ-compliant up to depth j, and that $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z^{(v)}), \pi^{(v)})$ accepts for all $v \in \mathsf{T}$, with probability $1 - 2j \cdot \kappa(\lambda)$.

For $j = 0$ the statement is implied by $\mathbb{V}$ accepting.

Now suppose that $\mathsf{T}_{j-1} \leftarrow \mathbb{E}_{j-1}$ is Φ-compliant up to depth $j-1$, and that $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z^{(v)}), \pi^{(v)})$ accepts for all $v \in \mathsf{T}_{j-1}$, with probability $1-2(j-1)\cdot\kappa(\lambda)$. Let $(\vec{\mathbb{i}}, (\mathsf{ivk}_v, z^{(v)})_v, (\pi^{(v)})_v, (\Phi, \mathsf{T}'), \vec{\mathbb{w}}) \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}_j}$.

By knowledge soundness, with probability $1 - 2\kappa(\lambda)$:

- for every vertex $v \in l_{\mathsf{T}'}(j)$, $(R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}, (\mathsf{ivk}_v, z^{(v)}), \mathbb{w}^{(v)}) \in \mathcal{R}_{\mathrm{R1CS}}$,
- by induction T' is Φ-compliant up to depth $j - 1$, and
- for $v \in l_{\mathsf{T}'}(j)$, v is labeled with $(z^{(v)}, \pi^{(v)})$ and $\mathsf{ivk}_v = \mathsf{ivk}$ where $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs}, \Phi)$.

Here we use the auxiliary output in the knowledge soundness definition of ARG to ensure consistency between the values $z^{(v)}$ and T', and to ensure that T' is Φ-compliant.

Now since $(R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}, (\mathsf{ivk}_v, z^{(v)}), \mathbb{w}^{(v)}) \in \mathcal{R}_{\mathrm{R1CS}}$, we obtain from $\mathbb{w}^{(v)}$ either

- local data $z^{(v)}_{\mathsf{loc}}$ and input messages $\left(z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}}\right)_{i\in[m]}$ such that $\Phi(z^{(v)}, z^{(v)}_{\mathsf{loc}}, z^{(1)}_{\mathsf{in}}, \ldots, z^{(m)}_{\mathsf{in}})$ accepts and for all $i \in [m]$ the SNARK verifier $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z^{(i)}_{\mathsf{in}}), \pi^{(i)}_{\mathsf{in}})$ accepts; or
- local data $z^{(v)}_{\mathsf{loc}}$ such that $\Phi(z^{(v)}, z^{(v)}_{\mathsf{loc}}, \bot, \ldots, \bot)$ accepts.

In both cases we append $z^{(v)}_{\mathsf{loc}}$ to the label of v. In the former case we label the children of v with $(z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}})$, and so v is Φ-compliant, and all of its descendants w have that $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z^{(w)}), \pi^{(w)})$ accepts. In the latter case, v has no children and so is Φ-compliant by the base case condition.

Hence by induction, $(\Phi, \mathsf{o}, \pi, \mathsf{T}) \leftarrow \mathbb{E}$ has Φ-compliant T, and $\mathsf{o}(\mathsf{T}) = \mathsf{o}$ by construction.

Since (Φ, o, π) are "passed up" from $\tilde{\mathbb{P}}$ via a series of d extractors, the distribution output by $\mathbb{E}$ is $d \cdot \kappa(\lambda)$-close, in statistical distance, to the output of $\tilde{\mathbb{P}}$ by the knowledge soundness of ARG.

11.5.2 Post-quantum security

In the quantum setting, $\tilde{\mathbb{P}}$ is taken to be a polynomial-size *quantum* circuit; hence also $\tilde{\mathcal{P}}_j, \mathcal{E}_{\tilde{\mathcal{P}}_j}, \mathbb{E}_j$ are quantum circuits for all j, as is the final extractor $\mathbb{E}$. Our definition of knowledge soundness is such that this proof then generalizes immediately to show security against quantum adversaries. In particular, the only difficulty arising from quantum adversaries is that they can generate their own randomness, whereas in the classical case we can force an adversary to behave deterministically by fixing its randomness. This is accounted for by the distributional requirement placed on the extractor of the argument system ARG.

11.5.3 Zero knowledge

Zero knowledge follows immediately from the zero knowledge guarantee of ARG, applied to the honest adversary $\mathcal{A}'$ which runs the honest PCD adversary $(\Phi, \mathsf{o}) \leftarrow \mathcal{A}(\mathsf{urs})$ and outputs $(R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}, (\mathsf{ivk}, \mathsf{o}))$ where $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs}, R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}})$. Note that here quantum adversaries are irrelevant since we are considering statistical zero knowledge.

12 Implementation of recursive composition

In Section 12.1 we describe our implementation work to realize our preprocessing zkSNARK (FRACTAL), and in Section 12.2 we describe our implementation work to realize recursive composition.

12.1 The preprocessing zkSNARK

Our starting point is `libiop` [SCI19], a library that provides (a) an implementation of the BCS transformation, which compiles any public-coin IOP into a corresponding SNARG by using (instantiated) random oracles; and (b) the *non*-holographic IOPs for R1CS underlying Aurora [BCRSVW19] and Ligero [AHIV17].

Our work to implement FRACTAL consists of (1) extending the BCS transformation to compile any public-coin *holographic* IOP into a corresponding *preprocessing* SNARG (following Section 10); and (2) implementing our efficient holographic IOP for R1CS (following Section 8). We discuss each in turn.

(1) From holography to preprocessing. Our transformation from Section 10 is a black-box extension of the BCS transformation (see Construction 10.3), which made it possible to extend the current implementation of the BCS transformation while re-using much of the existing infrastructure. We modified the generic IOP infrastructure in `libiop` to additionally support expressing *holographic* IOPs, by providing an indexer algorithm (in addition to the prover and verifier algorithms). We modified the transformation to determine if the input IOP is holographic and, if so, to additionally produce an indexer for the argument system, which uses a Merkle tree on the encoded index to produce an index proving key and index verification key. In this case, the prover and verifier for the argument use these keys to produce/authenticate answers about the encoded index, following our construction. Overall, our implementation simultaneously supports the old transformation (from IOP to SNARG) and our new one (from holographic IOP to preprocessing SNARG).

(2) Holographic IOP for R1CS. Our holographic IOP is built from two components (see Theorem 8.2): an RS-encoded holographic IOP and a low-degree test. For the latter, we reuse the generic low-degree testing infrastructure in `libiop`: the randomized reduction from testing multiple words to testing single words, and the FRI low-degree test [BBHR18]. Our implementation work is about the former component.

Specifically we implement the RS-encoded holographic IOP summarized in Fig. 4 (or, more precisely, an optimized and parametrized refinement of it), along with its indexer algorithm (not part of the figure). We reuse the reduction from R1CS to lincheck from the Aurora protocol in `libiop` (as our protocol shares the same reduction). The new key component that we implement is a holographic *multi*-lincheck, which simultaneously supports checking multiple linear relations that were holographically encoded. We believed that this building block of the protocol is of independent interest for the design of holographic proofs.

In addition to enabling sublinear verification, the holographic setting also presents new opportunities for improvements in the concrete efficiency of certain subroutines of the verifier, because we can use the indexer to provide useful precomputed information to the verifier. We leverage such opportunities to precompute various algebraic objects (such as vanishing polynomials), achieving noticeable efficiency improvements.

12.2 Designing the verifier's constraint system

In order to recursively compose FRACTAL, we need to design a constraint system that expresses its verifier. We describe a *general* method for designing constraint systems for the verifiers of SNARGs obtained by combining an RS-encoded IOP and the FRI low-degree test (as in Theorem 8.2) and then transforming the resulting IOP into a SNARG using Theorem 10.1 (henceforth referred to as the "BCS transformation").

The verifier in such SNARGs splits naturally into an "algebraic" part arising from the underlying IOP (hereafter the "IOP verifier") and a "hash-based" part arising from the BCS transformation (described in

Construction 10.2, hereafter the "BCS verifier"). We treat them separately: the BCS verifier is discussed in Section 12.2.1 and the IOP verifier in Section 12.2.2.

12.2.1 The BCS verifier

The BCS verifier can be further broken down into two subcomponents. The first is a hashchain that ensures that the IOP verifier's randomness for each round is correctly derived from the Merkle roots (Steps 1 and 2 of the BCS verifier in Construction 10.2). The second is the verification of the Merkle tree authentication paths to ensure the validity of the query answers (Step 3 of the BCS verifier in Construction 10.2).

The hashchain. The hashchain computation of the BCS verifier is as follows: intialize $\sigma_0 := \mathsf{ivk}\|\mathbb{x}$; then, for each round $i \in \{1, \ldots, \mathsf{k}\}$, derive the randomness $\rho_i := \rho(\sigma_{i-1})$ and use the i-th root rt_i in the argument π to compute $\sigma_i := \rho(\sigma_{i-1}\|\mathsf{rt}_i)$; finally, derive the post-interaction randomness $\rho_{\mathsf{k}+1} := \rho(\sigma_{\mathsf{k}})$. We require a constraint system $\mathcal{S}$ that, given assignments to the variables $(\mathsf{ivk}, \mathbb{x}, \mathsf{rt}_1, \rho_1, \ldots, \mathsf{rt}_{\mathsf{k}}, \rho_{\mathsf{k}})$, is satisfiable if and only if these assignments are consistent with this hashchain computation.

We realize $\mathcal{S}$ via a sponge construction [BDPV08], where first ivk and $\mathbb{x}$ are absorbed into the state and then, for each round $i \in \{1, \ldots, \mathsf{k}\}$, the randomness ρ_i is squeezed from the state and the i-th root rt_i is absorbed into the state; the post-interaction randomness $\rho_{\mathsf{k}+1}$ is then squeezed from the state. See Fig. 5 for a diagram of this. The size of the constraint system $\mathcal{S}$ is

$$S_{\mathsf{in}}(|\mathsf{ivk}| + |\mathbb{x}|) + \left(\sum_{i=1}^{\mathsf{k}} S_{\mathsf{in}}(|\mathsf{rt}_i|) + S_{\mathsf{out}}(|\rho_i|)\right) + S_{\mathsf{out}}(|\rho_{\mathsf{k}+1}|)$$

where $S_{\mathsf{in}}(n)$ denotes the number of constraints to absorb n field elements and $S_{\mathsf{out}}(n)$ denotes the number of constraints to squeeze n field elements. Naturally, these numbers depend on the particular choice of state transformation that is used to instantiate the sponge (see our evaluation in Section 13.2).

The above discussion omits some details. First, in some rounds the prover sends auxiliary information beyond the Merkle root (e.g., the third message of the prover in Fig. 4 includes a field element that allegedly equals an evaluation of the polynomial $\hat{t}$), and this auxiliary information must be absorbed together with the Merkle root. Second, Fig. 5 suggests that the rate of the sponge is large enough to absorb/squeeze any round's root/randomness with a single application of the state transformation, but this need not be the case, especially if sizes vary across rounds (e.g., we expect $|\mathsf{ivk}| + |\mathbb{x}|$ to be larger than $|\mathsf{rt}_i|$). Indeed, in our implementation we pick the rate of the sponge in such a way as to minimize the overall number of constraints for the hashchain, which means that some information may be absorbed/squeezed across multiple applications of the state transformation.

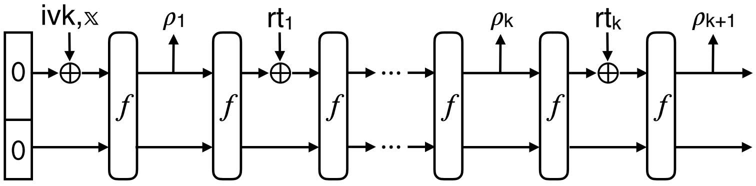

Figure 5: We use a sponge construction to realize the hashchain in the BCS verifier.

Authentication paths. For every query made by the IOP verifier to the encoded index or to a proof oracle, the BCS prover provides an authentication path for that query relative to the appropriate Merkle root. Recall that the index verification key ivk contains the root rt_0 of the Merkle tree on the encoded index $\mathbf{I}(\mathbb{i})$; and the argument π contains the roots $\mathsf{rt}_1, \dots, \mathsf{rt}_\mathsf{k}$ of the k Merkle trees that correspond to the k rounds of interaction. For every $i \in \{0, 1, \dots, \mathsf{k}\}$, we denote by Q_i the queries to the leaves of the i-th Merkle tree, by A_i the claimed query answers, and by W_i the corresponding auxiliary information to validate them. Both A_i and W_i are provided in the argument π for all i (including $i = 0$). Overall, we require a constraint system $\mathcal{S}$ that, given assignments to the variables $(\mathsf{rt}_i, Q_i, A_i, W_i)_{i=0}^{\mathsf{k}}$, is satisfiable if and only if, for every $i \in \{0, 1, \dots, \mathsf{k}\}$, the auxiliary information W_i validates the claimed answers in A_i with respect to the queries in Q_i.

Below we describe the basic approach to designing the constraint system. Afterward we describe how to significantly reduce the number of constraints via several optimizations.

The basic approach is to individually validate an authentication path for each query via a separate constraint system. Namely, let rt be a root, $j = \sum_{k=1}^{d} j_k 2^{k-1}$ a query location (in binary representation), a a claimed answer, s a salt used for hiding, and $(u_k)_{k=1}^{d}$ an authentication path. We require a constraint system that, given assignments to the variables $(\mathsf{rt}, j, a, s, (u_k)_{k=1}^{d})$, is satisfiable if and only if the check in the following computation passes: (1) let v_d be the hash of the salted answer $a \| s$; (2) for each $k = d, \dots, 1$: if the k-th bit of j is 0 then let v_{k-1} be the hash of $v_k \| u_k$, and if instead it is 1 then let v_{k-1} be the hash of $u_k \| v_k$; (3) check that $v_0 = \mathsf{rt}$. See Fig. 6 for a diagram of this constraint system.

If we denote by $S_{2\to1}$ the number of constraints to hash two hashes into a single hash, by S_{cswap} the number of constraints for a "controlled swap" on two hashes, and by $S_{\mathsf{leaf}}(n)$ the number of constraints to hash the answer and salt into a single hash, then the number of constraints for the above computation is

$$S_{\mathsf{leaf}}(|a| + |s|) + d \cdot (S_{\mathsf{cswap}} + S_{2\to1}) \enspace .$$

If we replicate the above strategy for each round $i \in \{0, 1, \dots, \mathsf{k}\}$ and each query in the query set Q_i, then the total number of constraints to validate all the query answers is:

$$\sum_{i=0}^{\mathsf{k}} |Q_i| \cdot \Big(S_{\mathsf{leaf}}(\alpha_i + \sigma_i) + d_i \cdot (S_{\mathsf{cswap}} + S_{2\to1}) \Big) \enspace , \tag{5}$$

where $\alpha_i \in \mathbb{N}$ denotes the number of field elements to answer a query in round i, $\sigma_i \in \mathbb{N}$ the number of field elements in a salt in round i, and $d_i \in \mathbb{N}$ the depth of the Merkle tree in round i. (Note that σ_0 is always 0 because no hiding is needed for the round that involves the encoded index; σ_i may also be 0 for $i > 0$ in some protocols because zero knowledge may not rely on any query bound to oracles in the i-th round.)

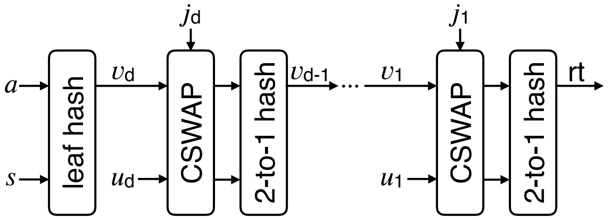

Figure 6: Diagram of a constraint system for validating an authentication path.

We can do significantly better than Eq. (5) if we leverage the structure of query sets, as we now describe.

First, there are known optimizations that increase "leaf size" to reduce argument size [BBHR19; BCRSVW19] that we use to also reduce the number of constraints in our setting. We explain these below.

- *Hash by column.* In protocols derived from RS-encoded IOPs using Theorem 8.2, each round's oracles are over the same domain and the IOP verifier queries the same locations across those oracles. This includes the “0-th round oracles”, i.e., the oracles in the encoded index $\mathbf{I}(\mathbb{i})$. Hence, for $i = 0, 1, \ldots, \mathsf{k}$, the BCS prover can hash the i-th round oracles *column-wise*: if $\ell_i \in \mathbb{N}$ is the number of oracles in the i-th round then each leaf in the i-th Merkle tree contains a vector in $\mathbb{F}^{\ell_i}$ (the “column”) representing one symbol from each of the ℓ_i oracles. Thus a single authentication path suffices to authenticate all the answers from a query to the entire leaf. This not only reduces argument size (fewer authentication paths are included in the argument) but also reduces the number of constraints (fewer authentication paths are validated).

- *Hash by subset.* The low-degree test that we use (see Section 12.2.2) yields queries that consider each domain as partitioned into subsets of equal size and each query requests the values of all locations in a subset, i.e., for each round i there is a parameter $m_i \in \mathbb{N}$ for which queries to oracles in the i-th round are always grouped in disjoint subsets of size m_i. Hence the BCS prover can hash all of these locations as part of the same leaf, which now is expanded from a vector in $\mathbb{F}^{\ell_i}$ to a matrix in $\mathbb{F}^{m_i \times \ell_i}$. This reduces the number of authentication paths, and also reduces the depth of the i-th Merkle tree by $\log_2 m_i$ levels.

Second, there are optimizations that pertain only to the goal of reducing the number of constraints, as we now exemplify. Since each oracle is queried at several locations, many authentication paths will overlap in the top layers of the Merkle trees. For argument size, this leads to the optimization of *path pruning* where the argument will contain the minimal collection of hashes that suffices to authenticate a *set* of queries. This optimization (which we continue to use for argument size) does *not* significantly reduce the number of constraints because validating the set of queries still involves re-computing the omitted hashes. Even worse, since query locations are random, we cannot hard-code in the constraint system which hash computations are repeated. We mitigate this problem via the following hybrid approach.

- *Tree cap.* By the pigeonhole principle, any set of authentication paths must overlap towards the top of the tree. To take advantage of this, we modify the Merkle tree in each round i by connecting the vertices at layer t_i (to be chosen later) directly to the root (and discarding the layers in between), so that the root has degree 2^{t_i}. We then compute the Merkle tree root using a “tree cap” hash function taking in 2^{t_i} hashes. Letting $S_{\mathsf{cap}}(n)$ denote the number of constraints for such a hash of n hashes, the total number of constraints across all rounds for the first layer alone is $\sum_{i=0}^{\mathsf{k}} S_{\mathsf{cap}}(2^{t_i})$.
- *Other layers.* For the other layers, we treat the authentication paths as disjoint, and allocate a separate constraint system to validate the segment of each such path. This amounts to invoking the basic strategy described above, whose cost is summarized in Eq. (5), with the modification that the authentication path is reduced from length d_i to length $d_i - t_i$. Note that, in light of the above discussions on answer size, we know that the answer in round i is of size $\alpha_i := m_i \cdot \ell_i$.

The above hybrid approach yields a total number of constraints equal to

$$\sum_{i=0}^{\mathsf{k}} S_{\mathsf{cap}}(2^{t_i}) + |Q_i| \cdot \Big(S_{\mathsf{leaf}}(m_i \cdot \ell_i + \sigma_i) + (d_i - t_i) \cdot (S_{\mathsf{cswap}} + S_{2\to 1})\Big) \ .$$

The constants t_i are chosen to minimize the above expression. In Section 13.2 we discuss the concrete improvements of the hybrid approach over the simplistic approach.

Remark 12.1 (arity of the Merkle tree)**.** There are algebraic hash functions for which using Merkle trees with large arity significantly reduces the number of constraints required to check many authentication paths [Alb+19a; GKKRRS19]. This comes at the cost of a larger argument size, and our implementation currently does not provide the option of such tradeoffs.

Representing query locations. We have so far assumed that the inputs and outputs of hash functions are *field elements*, as opposed to bits. This is because we instantiate all hash functions via algebraic constructions that require fewer constraints to express (see Section 13.2) and also because certain aspects of the verifier are simpler (e.g., the verifier's randomness in the protocol is essentially uniformly random in $\mathbb{F}$). That said, for the *query* part, the verifier does not draw query locations from the whole field $\mathbb{F}$ but, instead, from an evaluation domain contained in $\mathbb{F}$, and we need to obtain a bit-representation of these locations to check Merkle tree authentication paths. Recall also that queries are grouped into subset, and so the location will refer to the subset in the evaluation domain rather than to a single element in the evaluation domain.

We thus perform a bit decomposition of the field elements output by the hash function, split the resulting string into substrings of appropriate size, and regard each substring as a bit-representation of the queried subset. In more detail, for each round $i \in \{0, 1, \dots, \mathsf{k}\}$, let $L_i \subseteq \mathbb{F}$ be the evaluation domain of round i and recall that queries in round i are on subsets of size m_i. This means that we can obtain $\lfloor \log_2 |\mathbb{F}| \rfloor / (\log_2 |L_i| - \log_2 m_i)$ subsets in L_i for each element in $\mathbb{F}$ output by the hash function. Therefore, if we need to sample q subsets in L_i, the number of field elements that we need to allocate in $\rho_{\mathsf{k}+1}$ is $\lceil \mathsf{q} \cdot \frac{\log_2 |L_i| - \log_2 m_i}{\lfloor \log_2 |\mathbb{F}| \rfloor} \rceil$. Obtaining from these field elements the corresponding bit representations requires about $\mathsf{q} \cdot (\log_2 |L_i| - \log_2 m_i + 2)$ constraints.

We stress that queries across rounds need not be independent. Indeed, for the IOP verifiers that we consider (see Section 12.2.2), it will hold that each round receives the same number of queries (informally, there exists q such that $\mathsf{q} = |Q_0| = |Q_1| = \cdots = |Q_{\mathsf{k}}|$) and all the queries are correlated in that the q subsets for each round can all be derived from q samples of a certain length.

12.2.2 The IOP verifier

We describe the design of a constraint system that can express the verifier of any (holographic) IOP derived from an RS-encoded (holographic) IOP and a low-degree test, according to the construction underlying Theorem 8.2. Informally, the RS-encoded (holographic) IOP is an interactive reduction that leads to a set of algebraic claims about the prover's oracles (and possibly also about the encoded index); and the low-degree test is an interactive protocol that is used to ensure that these algebraic claims hold.

Outline. Let $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ be a holographic IOP for an indexed relation $\mathcal{R}$ constructed via Theorem 8.2. Note that $\mathcal{R}$ need not be the R1CS indexed relation. Below we recall the two ingredients of the construction.

- A $\mathsf{k}^{\mathcal{R}}$-round RS-encoded holographic IOP $(\mathbf{I}_{\mathcal{R}}, \mathbf{P}_{\mathcal{R}}, \mathbf{V}_{\mathcal{R}}, \{\vec{d}_{\mathbf{I}}, \vec{d}_{\mathbf{P},1}, \dots, \vec{d}_{\mathbf{P},\mathsf{k}^{\mathcal{R}}}\})$ over a domain L, with maximum degree $(d_{\mathsf{c}}, d_{\mathsf{e}})$, for the indexed relation $\mathcal{R}$. In each of $\mathsf{k}^{\mathcal{R}}$ rounds, the RS-hIOP verifier $\mathbf{V}_{\mathcal{R}}$ sends randomness and the RS-hIOP prover $\mathbf{P}_{\mathcal{R}}$ sends an oracle; after the interaction, the RS-hIOP verifier $\mathbf{V}_{\mathcal{R}}$ outputs a set of rational constraints (the algebraic claims, see Definition 4.1). We view the RS-hIOP verifier $\mathbf{V}_{\mathcal{R}}$ as a function that maps an instance $\mathbb{x}$ and randomness $\boldsymbol{\rho}^{\mathcal{R}}$ to a set of rational constrains $\mathcal{C}$.
- A $\mathsf{k}^{\mathrm{LDT}}$-round low-degree test $(\mathbf{P}_{\mathrm{LDT}}, \mathbf{V}_{\mathrm{LDT}})$ for the Reed–Solomon code $\mathrm{RS}[L, d_{\mathsf{c}}]$. In each of $\mathsf{k}^{\mathrm{LDT}}$ rounds, the LDT verifier $\mathbf{V}_{\mathrm{LDT}}$ sends randomness and the LDT prover $\mathbf{P}_{\mathrm{LDT}}$ sends an oracle; after the interaction, the LDT verifier $\mathbf{V}_{\mathrm{LDT}}$ makes $\mathsf{q}^{\mathrm{LDT}}$ queries, and then accepts or rejects. We can view the LDT verifier $\mathbf{V}_{\mathrm{LDT}}$ as two algorithms: a *query algorithm* $\mathbf{V}_{\mathrm{LDT}}.\mathrm{Q}$ such that $(Q_0, Q_1, \dots, Q_{\mathsf{k}^{\mathrm{LDT}}}) := \mathbf{V}_{\mathrm{LDT}}.\mathrm{Q}(\boldsymbol{\rho}^{\mathrm{LDT}})$ are the queries to the tested oracle and the $\mathsf{k}^{\mathrm{LDT}}$ prover oracles on randomness $\boldsymbol{\rho}^{\mathrm{LDT}}$; and a *decision algorithm* $\mathbf{V}_{\mathrm{LDT}}.\mathrm{D}$ such that $\mathbf{V}_{\mathrm{LDT}}.\mathrm{D}(A_0, A_1, \dots, A_{\mathsf{k}^{\mathrm{LDT}}}, \boldsymbol{\rho}^{\mathrm{LDT}})$ is the decision of the verifier given answers to the queries and the same randomness.

We need to design a constraint system to express the computation of the holographic IOP verifier $\mathbf{V}$, and so we are faced with three sub-tasks: (1) design a constraint system for the RS-hIOP verifier $\mathbf{V}_{\mathcal{R}}$; (2) design a constraint system for the LDT verifier $\mathbf{V}_{\mathrm{LDT}}$; (3) combine these two into a constraint system for $\mathbf{V}$.

We review features of the construction in Theorem 8.2 relevant for designing the constraint system.

- *Randomness.* The verifier $\mathbf{V}$ has $\mathsf{k}^{\mathcal{R}} + \mathsf{k}^{\mathrm{LDT}}$ rounds of interaction where the first $\mathsf{k}^{\mathcal{R}}$ rounds are for $\mathbf{V}_{\mathcal{R}}$ and the remaining $\mathsf{k}^{\mathrm{LDT}}$ rounds are for $\mathbf{V}_{\mathrm{LDT}}$. This means that we can split the randomness $\boldsymbol{\rho}$ of $\mathbf{V}$ into randomness $\boldsymbol{\rho}^{\mathcal{R}}$ for $\mathbf{V}_{\mathcal{R}}$ and randomness $\boldsymbol{\rho}^{\mathrm{LDT}}$ for $\mathbf{V}_{\mathrm{LDT}}$.

- *Domains.* The oracles in the encoded index and in the first $\mathsf{k}^{\mathcal{R}}$ rounds are all over the domain L, while oracles in the other $\mathsf{k}^{\mathrm{LDT}}$ rounds are over domains determined by the LDT.

- *Queries.* The queries to the oracles in the encoded index and in the first $\mathsf{k}^{\mathcal{R}}$ rounds are all the same, i.e., they are specified by the query set $Q_0 \subseteq L$ for the tested oracle determined by the LDT verifier.

- *Tested oracle.* The low-degree test is invoked on a "virtual oracle" $f \colon L \to \mathbb{F}$ defined as a random linear combination of rational constraints output by the RS-hIOP verifier. Namely, if $((p_k, q_k, d_k))_{k=1}^{r}$ are the rational constraints output by $\mathbf{V}_{\mathcal{R}}(\mathbb{x}; \boldsymbol{\rho}^{\mathcal{R}})$, $\alpha_1, \dots, \alpha_k$ are the random coefficients, and $f_1, \dots, f_\ell \colon L \to \mathbb{F}$ are the oracles sent by the RS-hIOP prover across the $\mathsf{k}^{\mathcal{R}}$ rounds, then f is defined as follows:

$$\forall\, a\,,\ f(a) := \sum_{k=1} \alpha_k \cdot \frac{p_k(a, f_1(a), \dots, f_\ell(a))}{q_k(a)} \quad .$$

  The low-degree test will read f at the query set Q_0, which means that all oracles $f_1, \dots, f_\ell$ will also be read at Q_0, and their answers must be combined according to the rational constraints and random coefficients.

(1) RS-hIOP. First we note that the structure of the interactive phase of the RS-hIOP for $\mathcal{R}$ determines what the hashchain described in Section 12.2.1 needs to squeeze and absorb for the first $\mathsf{k}^{\mathcal{R}}$ rounds. In the case of our RS-hIOP for R1CS this round information can be directly read off from Fig. 4.

We now turn to discussing the constraint system associated to $\mathbf{V}_{\mathcal{R}}$, which is tasked to evaluate the rational constraints output by $\mathbf{V}_{\mathcal{R}}$ at a set of locations $Q \subseteq L$ (the queries for the oracle tested by the LDT).

Suppose that the number of oracles sent by the RS-hIOP prover across the $\mathsf{k}^{\mathcal{R}}$ rounds is ℓ, and suppose that the number of rational constraints output by the RS-hIOP verifier is r. We seek a constraint system that, given as input an instance $\mathbb{x}$, randomness $\boldsymbol{\rho}^{\mathcal{R}}$, query set Q, answers from all oracles $(\beta_{a,j})_{a \in Q, j \in [\ell]}$ and claimed evaluations $(\gamma_{a,k})_{a \in Q, k \in [r]}$, is satisfiable if and only if, letting $((p_k, q_k, d_k))_{k=1}^{r}$ be the rational constraints output by $\mathbf{V}_{\mathcal{R}}(\mathbb{x}; \boldsymbol{\rho}^{\mathcal{R}})$, it holds that

$$\forall\, a \in Q\ ,\ \forall\, k \in [r]\ ,\ \gamma_{a,k} = \frac{p_k(a, \beta_{a,1}, \dots, \beta_{a,\ell})}{q_k(a)} \quad .$$

In all known RS-encoded protocols, including the RS-hIOP for R1CS in Fig. 4, the rational constraints output by the RS-hIOP verifier depend on the instance $\mathbb{x}$ and randomness $\boldsymbol{\rho}^{\mathcal{R}}$ in an algebraic way, in the sense that we can view $\mathbb{x}$ and $\boldsymbol{\rho}^{\mathcal{R}}$ as auxiliary variables of the arithmetic circuits $(p_k)_{k=1}^{r}$. This means that the cost to check all the equations above is

$$|Q| \cdot \left(\sum_{k=1}^{r} |p_k| + |q_k| + 1 \right) \quad ,$$

where $|p_k|$ and $|q_k|$ denote the sizes of the arithmetic circuits for p_k and q_k. (The additive 1 accounts for checking the equality given variables that contain the outputs of the two arithmetic circuits.) Note that these complexity measures are related to, but different from, the query evaluation time defined in Section 4.1: query evaluation time is a *uniform* complexity measure, whereas circuit size is *non-uniform.*

In our implementation we additionally reuse sub-computations across constraint systems and across evaluation points to reduce the size of the constraint system. For example, if $q_k = q_{k'}$ for distinct k, k' then we know that we only need to compute $q_k(a)$ once; similarly if p_k and $p_{k'}$ share sub-computations. One can verify that there are several such opportunities for the RS-hIOP for R1CS in Fig. 4.

(2) Low-degree test. The low-degree test that we use in this paper is FRI [BBHR18], which is a logarithmic-round logarithmic-query protocol. Below we describe a constraint system that represents the FRI verifier.

Let L be the domain of the oracle to be tested (i.e., the domain of the RS-hIOP). The size of L induces a list of "localization parameters" $(\eta_1, \dots, \eta_{\mathsf{k}^{\mathrm{LDT}}})$ which in turn induces a list of domains $(L_1, \dots, L_{\mathsf{k}^{\mathrm{LDT}}})$ with progressively smaller sizes, $|L_i| = |L_{i-1}|/2^{\eta_i} = |L|/2^{\eta_1+\cdots+\eta_i}$ with $L_0 := L$. Each domain L_i is obtained from L_{i-1} as the image of a 2^{η_i}-to-1 map h_i that maps cosets of size 2^{η_i} in L_{i-1} to single points in L_i. Any coset U_0 of size 2^{η_1} in the domain $L_0 = L$ determines $\mathsf{k}^{\mathrm{LDT}} - 1$ cosets $(U_1, \dots, U_{\mathsf{k}^{\mathrm{LDT}}-1})$ of respective sizes $(2^{\eta_2}, \dots, 2^{\mathsf{k}^{\mathrm{LDT}}})$ contained in $(L_1, \dots, L_{\mathsf{k}^{\mathrm{LDT}}-1})$ as follows: for each $i \in \{1, \dots, \mathsf{k}^{\mathrm{LDT}} - 1\}$, U_i is the unique coset of size $2^{\eta_{i+1}}$ that contains the point $h_i(U_{i-1})$.

We separately address the interactive phase and the query phase.

- *Interactive phase.* For $i \in \{1, \dots, \mathsf{k}^{\mathrm{LDT}}\}$, in round i the FRI verifier sends a random field element α_i and the FRI prover replies with an oracle $f_i \colon L_i \to \mathbb{F}$ if $i < \mathsf{k}^{\mathrm{LDT}}$, or with a (non-oracle) message containing the coefficients of a polynomial $\hat{f}_{\mathsf{k}^{\mathrm{LDT}}}(X)$ if $i = \mathsf{k}^{\mathrm{LDT}}$. If the degree to be tested is d then the degree of $\hat{f}_{\mathsf{k}^{\mathrm{LDT}}}(X)$ is $d_{\mathsf{k}^{\mathrm{LDT}}} := d/2^{\eta_1+\cdots+\eta_{\mathsf{k}^{\mathrm{LDT}}}}$. Queries to domain L_{i-1} are grouped in cosets of size 2^{η_i}. Hence larger localization parameters lead to fewer rounds, at the expense of querying larger cosets. (Choosing these parameters well is crucial to minimizing constraint complexity, as we discuss in Section 13.2.)

  Since the FRI verifier is public-coin, its interactive phase does not yield any special constraints. However the specifics of the interaction affect the hashchain described in Section 12.2.1, which is responsible to squeeze verifier randomness and absorb prover messages. We deduce that, for each round $i \in \{1, \dots, \mathsf{k}^{\mathrm{LDT}} - 1\}$: the hashchain is required to squeeze a single field element and then to absorb a single Merkle root, and the depth of the corresponding Merkle tree is $\log_2 |L| - (\eta_1 + \cdots + \eta_{i+1})$. In the last round ($i = \mathsf{k}^{\mathrm{LDT}}$), the hashchain is required to squeeze a single field element and then to absorb $d_{\mathsf{k}^{\mathrm{LDT}}} + 1$ field elements.

- *Query phase.* The FRI verifier repeats the following q times, for a number q that controls soundness error.

  - *Queries.* The FRI verifier samples a random coset U_0 of size 2^{η_1} in the domain $L_0 = L$, and reads the values of the oracle to be tested at U_0. The coset U_0 determines, for each $i \in \{1, \dots, \mathsf{k}^{\mathrm{LDT}} - 1\}$, a coset U_i of size $2^{\eta_{i+1}}$ in the domain L_i, and the FRI verifier reads the values of the oracle f_i in round i at U_i. Finally, the FRI verifier also reads all the $d_{\mathsf{k}^{\mathrm{LDT}}} + 1$ coefficients of the polynomial sent in round $\mathsf{k}^{\mathrm{LDT}}$.
  - *Decision.* For each $i \in \{0, 1, \dots, \mathsf{k}^{\mathrm{LDT}} - 1\}$, let $p_i(X)$ be the polynomial of degree less than $2^{\eta_{i+1}}$ that equals the interpolation of the values read for the i-th coset U_i. Let $\hat{f}_{\mathsf{k}^{\mathrm{LDT}}}(X)$ be the polynomial of degree $d_{\mathsf{k}^{\mathrm{LDT}}}$ sent by the prover in the last round (round $i = \mathsf{k}^{\mathrm{LDT}}$).

    The FRI verifier performs the following $\mathsf{k}^{\mathrm{LDT}}$ consistency checks: for each $i \in \{1, \dots, \mathsf{k}^{\mathrm{LDT}} - 1\}$, check that $p_{i-1}(\alpha_i) = f_i(h_i(U_{i-1}))$; also check that $p_{\mathsf{k}^{\mathrm{LDT}}-1}(\alpha_{\mathsf{k}^{\mathrm{LDT}}}) = \hat{f}_{\mathsf{k}^{\mathrm{LDT}}}(\alpha_{\mathsf{k}^{\mathrm{LDT}}})$.

  The implication of the first item above to the constraint system is that the hashchain described in Section 12.2.1 needs to squeeze enough field elements to determine q samples of starting cosets in the domain $L_0 = L$ (with each sample indexed in binary as already discussed). Moreover, the constraint system needs to check, for each starting coset U_0, that the remaining $\mathsf{k}^{\mathrm{LDT}} - 1$ cosets U_i are correctly chosen. For this, since h_i has degree 2^{η_i}, we need at most 2^{η_i} constraints. Hence, the total constraint cost for checking q lists of cosets is $\mathsf{q} \cdot \sum_{i=1}^{\mathsf{k}^{\mathrm{LDT}}} 2^{\eta_i}$. (In fact, we can avoid this cost altogether: by choosing an appropriate bit representation, we can obtain the bit decomposition of the index of coset U_i by truncating the bit decomposition of the index of coset U_{i-1}, in which case *no* constraints are needed.)

The implication of the second item above to the constraint system is that the FRI verifier, for each of q runs, needs to evaluate the interpolation of $\mathsf{k}^{\mathrm{LDT}}$ cosets at a single point and also evaluate the polynomial contained in the last message at a single point. This number of constraints for this is

$$\mathsf{q} \cdot \left(S_{\mathsf{eval}}\left(\frac{d}{2^{\eta_1 + \cdots + \eta_{\mathsf{k}^{\mathrm{LDT}}}}} \right) + \sum_{i=1}^{\mathsf{k}^{\mathrm{LDT}}} S_{\mathsf{lde}}(2^{\eta_i}) \right) \quad ,$$

where $S_{\mathsf{eval}}(n) = n$ is the number of constraints to evaluate a polynomial of degree n (say, via a constraint system that follows Horner's method), and $S_{\mathsf{lde}}(n) = 2n + O(\log n)$ is the number of constraints to evaluate at a single point the interpolation of a function defined over a size-n coset. We justify this latter cost below, because the design of the constraint system for interpolation requires some care.

Coset interpolation. We require a constraint system that, given the identifier of a coset S in a domain L, a function $f \colon S \to \mathbb{F}$, an evaluation point $\gamma \in \mathbb{F}$, and a claimed evaluation $v \in \mathbb{F}$, is satisfiable if and only if $v = \sum_{a \in S} f(a) L_{a,S}(\gamma)$, where $\{L_{a,S}(X)\}_{a \in S}$ are the Lagrange polynomials for S.

We describe a constraint system of size $2|S| + O(\log(|S|))$. Given the Lagrange coefficients, we can compute the inner product of the function and the Lagrange coefficients with $|S|$ constraints. This leaves $|S| + O(\log(|S|))$ constraints to compute the Lagrange coefficients, as we discuss below.

A simplistic approach would be to deduce the coefficients of each Lagrange polynomial $\{L_{a,S}(X)\}_{a \in S}$, hardcode these coefficients in the constraint system, and then let the constraint system compute $\{L_{a,S}(\gamma)\}_{a \in S}$ for the given evaluation point $\gamma \in \mathbb{F}$. However, the choice of coset S is *not* known at "compile time" (when constructing the constraint system) because the identifier of S in the domain L is an input to the constraint system. We now explain how to efficiently compute all the evaluations without "generically" deriving the coefficients of each Lagrange polynomial (which would be much more expensive).

Observe that, at compile time, we know *some* information about S: the *base coset* (i.e., subgroup) S^* from which the coset S is derived as a shift (S^* need not be in L). Namely, in the additive case $S = S^* + \xi$ for some $\xi \in \mathbb{F}$, and in the multiplicative case $S = \xi S^*$ for some $\xi \in \mathbb{F}$. Thus the identifier of S in L can be viewed as encoding the shift ξ that determines S from S^*. This is useful because: (a) the vanishing polynomial of a coset S is closely related to the vanishing polynomial of its base coset S^*; and (b) each Lagrange polynomial can be expressed via the vanishing polynomial $v_S(X)$ and its derivative $v'_S(X)$. Specifically, for every $a \in S$, $L_{a,S}(X) = \frac{1}{v'_S(a)} \cdot \frac{v_S(X)}{X-a}$. This enables us to hardcode in the constraint system information about the base coset S^*, and task the constraint system with a cheap computation that depends on the shift ξ.

We describe this approach for the additive and multiplicative cases separately.

- **Additive case.** The derivative $v'_S(X)$ is a constant $c_{S^*} \in \mathbb{F}$ that only depends on the base coset S^*. Hence all the values $\{v'_S(a)\}_{a \in S}$ (and their inverses) are known at compile time, as they all equal c_{S^*}. The polynomial $v_S(X)$ equals $v_{S^*}(X - \xi)$, which has $O(\log(|S|))$ non-zero monomials. Hence, if we hardcode the polynomial v_{S^*} in the constraint system, we can compute $v_S(\gamma) = v_{S^*}(\gamma - \xi) \in \mathbb{F}$, and also $v_S(\gamma)/c_{S^*} = v_{S^*}(\gamma - \xi)/c_{S^*} \in \mathbb{F}$ (common to all Lagrange coefficients) with $O(\log(|S|))$ constraints.

  Next, note that each element $a \in S$ can be written as $a = a^* + \xi$ for a corresponding element $a^* \in S^*$. This means that $\{X - a\}_{a \in S} = \{X - a^* - \xi\}_{a^* \in S^*}$, where the elements a^* are hardcoded in the constraint system and ξ is an input to the constraint system. In particular, given $v_{S^*}(\gamma - \xi)/c_{S^*} \in \mathbb{F}$ we can compute $\{L_{a,S}(\gamma)\}_{a \in S} = \{\frac{v_{S^*}(\gamma - \xi)}{c_{S^*}} \cdot \frac{1}{\gamma - a^* - \xi}\}_{a^* \in S^*}$ with $|S|$ additional constraints.

- **Multiplicative case.** The polynomial $v_S(X)$ is the polynomial $X^{|S|} - \xi^{|S|}$, and its derivative $v'_S(X)$ is the polynomial $|S|X^{|S|-1}$; recall that $|S| = |S^*|$ and so this quantity is known at compile time. Moreover, each element $a \in S$ can be written as $a = \xi a^*$ for a corresponding element $a^* \in S^*$. Therefore we can re-write each Lagrange polynomial as:

$$\begin{aligned} L_{a,S}(X) &= \frac{1}{v'_S(a)} \cdot \frac{v_S(X)}{X - a} \\ &= \frac{1}{|S|(\xi a^*)^{|S|-1}} \cdot \frac{X^{|S|} - \xi^{|S|}}{X - \xi a^*} \\ &= \frac{1}{|S|(\xi a^*)^{|S|}} \cdot \frac{X^{|S|} - \xi^{|S|}}{X(\xi a^*)^{-1} - 1} \\ &= \frac{1}{|S|\xi^{|S|}} \cdot \frac{X^{|S|} - \xi^{|S|}}{X(\xi a^*)^{-1} - 1} \quad . \end{aligned}$$

The above expression leads to the following strategy. The constraint system first uses the shift ξ and evaluation point γ to compute, via $O(\log(|S|))$ constraints, the value $\frac{\gamma^{|S|}-\xi^{|S|}}{|S|\xi^{|S|}}$; and also one constraint to compute $\gamma\xi^{-1}$. Then, the constraint system computes the values $\{L_{a,S}(\gamma)\}_{a \in S} = \{\frac{\gamma^{|S|}-\xi^{|S|}}{|S|\xi^{|S|}} \cdot \frac{1}{\gamma(\xi a^*)^{-1}-1}\}_{a^* \in S^*}$ with $|S|$ additional constraints.

13 Evaluation

In Section 13.1 we evaluate our implementation of the preprocessing zkSNARK, and in Section 13.2 we evaluate our implementation of recursive composition.

All reported measurements were run in single-threaded mode on a machine with an Intel Xeon 6136 CPU at 3.0 GHz with 252 GB of RAM (no more than 32 GB of RAM were used in any experiment).

13.1 Performance of the preprocessing zkSNARK

We report on the performance of FRACTAL, the preprocessing zkSNARK for R1CS that we have implemented by extending `libiop` as described in Section 12.1. We configure our implementation to achieve 128 bits of security, for constraints expressed over a prime field of 181 bits. This field choice is illustrative, as the only requirement on the field is that it should contain suitable subgroups for us to use.

In Fig. 8 we report the costs for several efficiency measures, and for each measure also indicate how much of the cost is due to the probabilistic proof and how much is due to the cryptographic compiler. The costs depend on the number of constraints n in the R1CS instance,[13] and so we report how the costs change as we vary n over the range $\{2^{10}, 2^{11}, \ldots, 2^{20}\}$. Below, by *native execution time* we mean the time that it takes to check that an assignment satisfies the constraint system, and by *native witness size* we mean the number of bytes required to represent an assignment to the constraint system.

- *Indexer time.* In the upper left, we plot the running time of the indexer, as absolute cost (top graph) and as relative cost when compared to native execution time (bottom graph). Indexer times range from fractions of a second to several minutes, and the plot confirms the quasilinear complexity of the indexer. Indexer time is dominated by the cost of running the underlying HIOP indexer.

- *Prover time.* In the upper right, we plot the running time of the prover, as absolute cost (top graph) and as relative cost when compared to native execution time (bottom graph). Prover times range from fractions of a second to several minutes, and the plot confirms the quasilinear complexity of the prover. Prover time is dominated by the cost of running the underlying HIOP prover.

- *Argument size.* In the lower left, we plot argument size, as absolute cost (top graph) and as relative cost when compared to native witness size (bottom graph). Argument sizes range from 80 kB to 200 kB with compression (argument size is smaller than native witness size) occurring for $n \geq 8,000$, and the plot confirms the polylogarithmic complexity of the argument. Argument size is dominated by the cryptographic digests to authenticate query answers.

- *Verifier time.* In the lower right, we plot the running time of the verifier, as absolute cost (top graph) and as relative cost when compared to native execution time (bottom graph). Verifier times are several milliseconds and become faster than native execution for $n \geq 65,000$, and the plot confirms the polylogarithmic complexity of the verifier. Verifier time is dominated by the cost of running the underlying HIOP verifier.

Finally, in Fig. 9, we compare FRACTAL with the state of the art in several types of zkSNARKs for R1CS:

[13] More precisely, the costs in general depend on (a) n, the number of constraints (i.e., number of rows in each matrix); (b) n', the number of variables (i.e., number of columns in each matrix); (c) m, the number of non-zero entries in a matrix; and (d) k, the number of public inputs. The number of constraints n and the number of variables n' are typically approximately equal, and indeed in this paper we have assumed for simplicity that $n = n'$ (the matrices in Definition 3.2 are square); so we only keep track of n. The number of non-zero entries m is typically within a small factor of n, and in our experiments m/n is approximately 1. Finally, the number of public inputs k is at most n', and in typical applications it is much smaller than n', so we do not focus on it.

(1) *Aurora*, a non-preprocessing zkSNARK in the (quantum) random oracle model [BCRSVW19];
(2) *Groth16*, a preprocessing zkSNARK with circuit-specific SRS [Gro16];
(3) *Marlin*, a preprocessing zkSNARK with universal SRS [CHMMVW19].
The first protocol is configured the same as our protocol (128 bits of security over a prime field of 181 bits), and the implementation that we use is from `libiop` [SCI19]. The second and third protocols require a choice of pairing-friendly elliptic curve, which we take to be `bls12-381`; the implementation of the second protocol is from `bellman` [bell15] and the implementation of the third protocol is from `marlin` [mar19].

13.2 Performance of recursive composition

We report on the performance of recursive composition based on FRACTAL, by discussing *verifier size*: the number of constraints to express the verifier's computation. This metric is significantly affected by the choice of hash function to instantiate the random oracle. So we first discuss a representative choice of hash function, and after that discuss verifier size.

A choice of hash function. In the evaluation of this section, we choose to instantiate the various hash functions introduced in Section 12.2.1 via *Rescue* [AABSDS19], which is a sponge hash function [BDPV08]. This means that the hash function maintains a state that is split into two parts: the *rate* part of the state, which is used to absorb inputs and squeeze outputs; and the *capacity* part of the state, which is only used for security purposes. The cost of absorbing and squeezing in Rescue is linear in its state size. We choose rate and capacity for all of the different hash functions of Section 12.2.1, by picking the smallest rate and capacity parameters that suffice to absorb the relevant input or squeeze the relevant output in a single application of Rescue.[14] In line with minimizing capacity, here we assume for simplicity of exposition that $\lfloor \log(|\mathbb{F}|) \rfloor \geq 2\lambda$, which allows us to set the capacity to be one field element, and the hash output size as one field element;[15] moreover, in our experiments, we set the α parameter of Rescue (which determines the S-box permutations) to 3 (as it is co-prime with the size of the multiplicative subgroup of the field in the experiments). Using the above ideas, we summarize the rate, capacity, and constraint complexities for the various hash functions in Fig. 7 in the case of 128 bits of security.

hash type	notation	rate	capacity	rounds	number of constraints
absorb	$S_{\mathsf{in}}(n)$	1	1	64	$256 \cdot n$
squeeze	$S_{\mathsf{out}}(n)$	1	1	64	$256 \cdot n$
leaf hash	$S_{\mathsf{leaf}}(n)$	$n \geq 6$	1	10	$40 \cdot (n+1)$
		$n < 6$	1	$2\lceil \frac{32}{n+1} \rceil$	$8 \cdot \lceil \frac{32}{n+1} \rceil \cdot (n+1)$
2-to-1 hash	$S_{2 \to 1}$	2	2 †	16	256
cap hash	$S_{\mathsf{cap}}(2^t), t > 2$	2^t	1	10	$40 \cdot (2^t + 1)$

Figure 7: Parameterization of Rescue to realize each hash function from Section 12.2.1, along with respective costs as number of constraints. The constraint complexity of a single Rescue sponge permutation is $4 \cdot \text{state-size} \cdot \text{rounds}$. The number of rounds is $\max(10, 2 \cdot \lceil \frac{\lambda}{4 \cdot \text{state size}} \rceil)$ (†: Due to rounding effects for small input sizes, it is more efficient to set 2 here.)

Verifier size. The size of the verifier as a circuit (or constraint system) determines the "minimal payload"

[14] The alternative of using a smaller rate and rely on multiple applications of Rescue would increase circuit size, because each application would incur the costs for the capacity.

[15] For smaller fields, we accordingly increase the rate and capacity (which in turn moderately increases the number of constraints).

that needs to be proved in each recursion. For example, in the construction of PCD described in Section 11, if one wishes to recursively check a given compliance predicate Φ taking m prior messages then (informally) one has to prove the satisfiability of a circuit that contains Φ plus m copies of the verifier. Therefore, the smaller the verifier circuit, the better. Recall from Definition 11.4 that $\mathcal{V}^{(\lambda,N,k)}$ denotes a circuit corresponding to the computation of the SNARK verifier $\mathcal{V}$, for security parameter λ, when checking indices of size at most N and instances of size at most k. Our goal is to minimize the size of this circuit. We shall fix the security parameter to be 128 bits, so we are only left with two parameters.

We have described our design of a verifier circuit in Section 12.2, leaving several parameters as unspecified (e.g., the number of commitments sent by the prover in a particular round, the number of field elements sent by the verifier in a particular round, the specific rational constraints, and so on). We now specialize this design for the verifier for FRACTAL, and then assemble these expressions in order to derive the cost of the verifier as the following cost model (which we have experimentally validated in our circuit writing):

$$2644 \cdot \log(N)^2 + 19058 \cdot \log(N) + 368 \cdot k + 94432 \ \ .$$

In Fig. 10 we plot the measured number of constraints in the verifier together with the number of constraints that the verifier is checking. The graph shows that for all computations of more than 2 million constraints, the number of constraints of the verifier *smaller* than the number of constraints that it is checking. This demonstrates feasibility of recursion for our implementation.

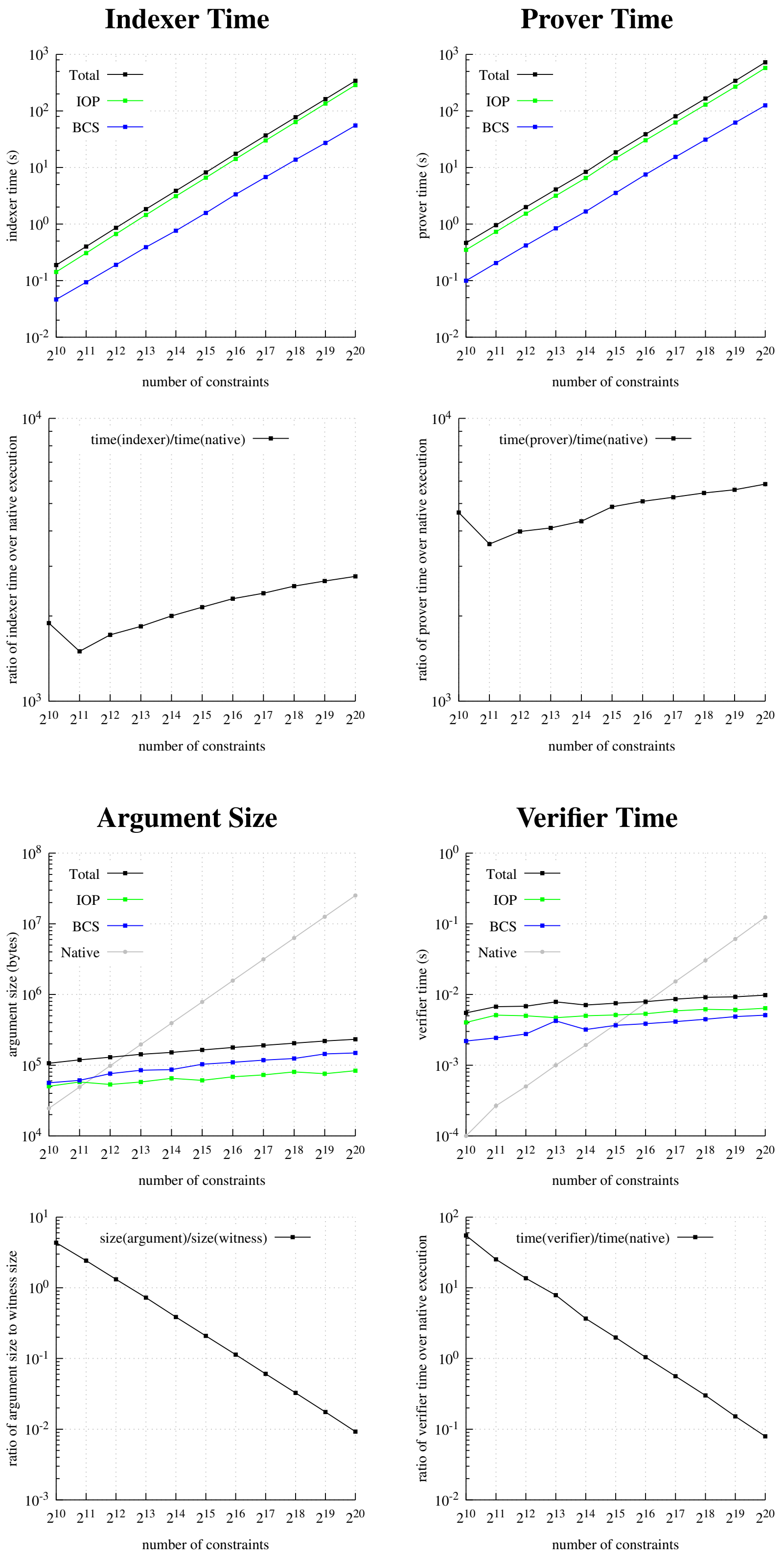

Figure 8: Performance of FRACTAL.

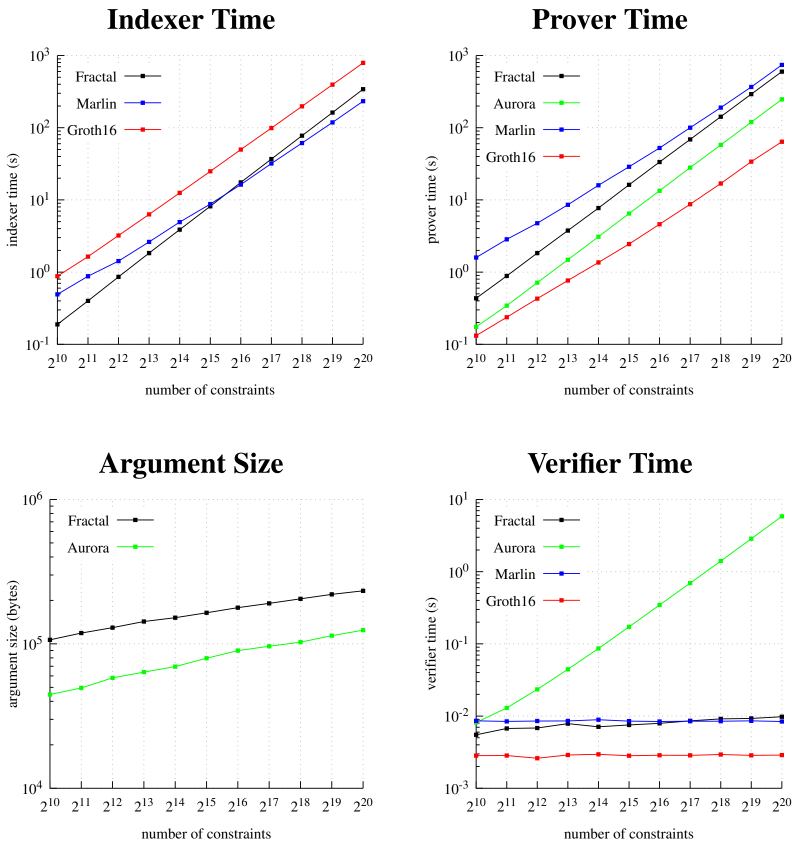

Figure 9: Comparison across several zkSNARKs for R1CS. The argument size for [Gro16] is 192 B and for [CHMMVW19, Marlin] is 1296 B; they are not plotted in the argument size graph because they are *much* smaller than the argument sizes for the other protocols (which differ in that they are post-quantum and transparent). Note that the setup algorithm for [Gro16] is plotted in the indexer graph because it also serves as an indexer.

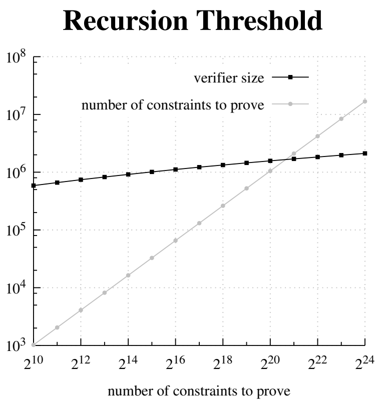

Figure 10: The polylogarithmic verifier size becomes smaller than the computation size that it is checking.

Acknowledgments

This research was supported in part by: the Berkeley Haas Blockchain Initiative, a Google Faculty Award, and donations from the Ethereum Foundation and the Interchain Foundation.

References

[AABSDS19] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. *Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols*. IACR Cryptology ePrint Archive, Report 2019/426. 2019.

[ABLSZ19] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, Janno Siim, and Michal Zajac. “UC-Secure CRS Generation for SNARKs”. In: *Proceedings of the 11th International Conference on Cryptology in Africa*. AFRICACRYPT ’19. 2019, pp. 99–117.

[AD18] Tomer Ashur and Siemen Dhooghe. *MARVELlous: a STARK-Friendly Family of Cryptographic Primitives*. IACR Cryptology ePrint Archive, Report 2018/1098. 2018.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. “Ligero: Lightweight Sublinear Arguments Without a Trusted Setup”. In: *Proceedings of the 24th ACM Conference on Computer and Communications Security*. CCS ’17. 2017, pp. 2087–2104.

[AS04] Scott Aaronson and Yaoyun Shi. “Quantum lower bounds for the collision and the element distinctness problems”. In: *Journal of the ACM* 51.4 (2004), pp. 595–605.

[Alb+19a] Martin R Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. *Algebraic Cryptanalysis of STARK-Friendly Designs: Application to MARVELlous and MiMC*. IACR Cryptology ePrint Archive, Report 2019/419. 2019.

[Alb+19b] Martin R Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. *Feistel Structures for MPC, and More*. IACR Cryptology ePrint Archive, Report 2019/397. 2019.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. “Fast Reed–Solomon Interactive Oracle Proofs of Proximity”. In: *Proceedings of the 45th International Colloquium on Automata, Languages and Programming*. ICALP ’18. 2018, 14:1–14:17.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. “Scalable Zero Knowledge with No Trusted Setup”. In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO ’19. 2019, pp. 733–764.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. “Recursive Composition and Bootstrapping for SNARKs and Proof-Carrying Data”. In: *Proceedings of the 45th ACM Symposium on the Theory of Computing*. STOC ’13. 2013, pp. 111–120.

[BCGGRS19] Eli Ben-Sasson, Alessandro Chiesa, Lior Goldberg, Tom Gur, Michael Riabzev, and Nicholas Spooner. “Linear-Size Constant-Query IOPs for Delegating Computation”. In: *Proceedings of the 17th Theory of Cryptography Conference*. TCC ’19. 2019.

[BCGTV15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. “Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs”. In: *Proceedings of the 36th IEEE Symposium on Security and Privacy*. S&P ’15. 2015, pp. 287–304.

[BCIOP13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. “Succinct Non-Interactive Arguments via Linear Interactive Proofs”. In: *Proceedings of the 10th Theory of Cryptography Conference*. TCC ’13. 2013, pp. 315–333.

[BCRSVW19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '19. Full version available at `https://eprint.iacr.org/2018/828`. 2019, pp. 103–128.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC '16-B. 2016, pp. 31–60.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Scalable Zero Knowledge via Cycles of Elliptic Curves". In: *Proceedings of the 34th Annual International Cryptology Conference*. CRYPTO '14. Extended version at `http://eprint.iacr.org/2014/595`. 2014, pp. 276–294.

[BDFLSZ11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. "Random Oracles in a Quantum World". In: *Proceedings of the 17th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '11. 2011, pp. 41–69.

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "On the Indifferentiability of the Sponge Construction". In: *Proceedings of the 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '08. 2008, pp. 181–197.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. "Checking computations in polylogarithmic time". In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*. STOC '91. 1991, pp. 21–32.

[BGG17] Sean Bowe, Ariel Gabizon, and Matthew Green. *A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK*. Cryptology ePrint Archive, Report 2017/602. 2017.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Halo: Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. 2019.

[BGHSV06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. "Robust PCPs of Proximity, Shorter PCPs, and Applications to Coding". In: *SIAM Journal on Computing* 36.4 (2006), pp. 889–974.

[BGKS19] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. *DEEP-FRI: Sampling Outside the Box Improves Soundness*. ECCC TR19-044. 2019.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. *Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model*. Cryptology ePrint Archive, Report 2017/1050. 2017.

[BR93] Mihir Bellare and Phillip Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS '93. 1993, pp. 62–73.

[CCHLRR18] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, and Ron D. Rothblum. *Fiat–Shamir From Simpler Assumptions*. Cryptology ePrint Archive, Report 2018/1004. 2018.

[CCRR92] Richard Chang, Suresh Chari, Desh Ranjan, and Pankaj Rohatgi. "Relativization: a revisionistic retrospective". In: *Bulletin of the European Association for Theoretical Computer Science* 47 (1992), pp. 144–153.

[CCW19] Alessandro Chiesa, Lynn Chua, and Matthew Weidner. "On Cycles of Pairing-Friendly Elliptic Curves". In: *SIAM Journal on Applied Algebra and Geometry* 3.2 (2019). `https://arxiv.org/abs/1803.02067`, pp. 175–192.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. "The random oracle methodology, revisited". In: *Journal of the ACM* 51.4 (2004), pp. 557–594.

[CHMMVW19] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. *Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS*. Cryptology ePrint Archive, Report 2019/1047. 2019.

[CMS19] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. "Succinct Arguments in the Quantum Random Oracle Model". In: *Proceedings of the 17th Theory of Cryptography Conference*. TCC '19. Available as Cryptology ePrint Archive, Report 2019/834. 2019.

[CT10] Alessandro Chiesa and Eran Tromer. "Proof-Carrying Data and Hearsay Arguments from Signature Cards". In: *Proceedings of the 1st Symposium on Innovations in Computer Science*. ICS '10. 2010, pp. 310–331.

[CTV15] Alessandro Chiesa, Eran Tromer, and Madars Virza. "Cluster Computing in Zero Knowledge". In: *Proceedings of the 34th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '15. 2015, pp. 371–403.

[Co17] O(1) Labs. *Coda Cryptocurrency*. `https://codaprotocol.com/`. 2017.

[DR04] Irit Dinur and Omer Reingold. "Assignment Testers: Towards a Combinatorial Proof of the PCP Theorem". In: *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '04. 2004, pp. 155–164.

[EKR04] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. "Fast approximate probabilistically checkable proofs". In: *Information and Computation* 189.2 (2004), pp. 135–159.

[FST10] David Freeman, Michael Scott, and Edlyn Teske. "A Taxonomy of Pairing-Friendly Elliptic Curves". In: *Journal of Cryptology* 23.2 (2010), pp. 224–280.

[For94] Lance Fortnow. "The Role of Relativization in Complexity Theory". In: *Bulletin of the European Association for Theoretical Computer Science* 52 (1994), pp. 229–244.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. "Quadratic Span Programs and Succinct NIZKs without PCPs". In: *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '13. 2013, pp. 626–645.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. "On the (In)security of the Fiat-Shamir Paradigm". In: *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '03. 2003, pp. 102–113.

[GKKRRS19] Lorenzo Grassi, Daniel Kales, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. *Starkad and Poseidon: New Hash Functions for Zero Knowledge Proof Systems*. IACR Cryptology ePrint Archive, Report 2019/458. 2019.

[GKMMM18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. "Updatable and Universal Common Reference Strings with Applications to zk-SNARKs". In: *Proceedings of the 38th Annual International Cryptology Conference*. CRYPTO '18. 2018, pp. 698–728.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *Journal of the ACM* 62.4 (2015), 27:1–27:64.

[GR15] Tom Gur and Ron D. Rothblum. "Non-Interactive Proofs of Proximity". In: *Proceedings of the 6th Innovations in Theoretical Computer Science Conference*. ITCS '15. 2015, pp. 133–142.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Report 2019/953. 2019.

[Gro10] Jens Groth. "Short Pairing-Based Non-interactive Zero-Knowledge Arguments". In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '10. 2010, pp. 321–340.

[Gro16] Jens Groth. “On the Size of Pairing-Based Non-interactive Arguments”. In: *Proceedings of the 35th Annual International Conference on Theory and Applications of Cryptographic Techniques*. EUROCRYPT ’16. 2016, pp. 305–326.

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

[Lip12] Helger Lipmaa. “Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments”. In: *Proceedings of the 9th Theory of Cryptography Conference on Theory of Cryptography*. TCC ’12. 2012, pp. 169–189.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. *Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updateable Structured Reference Strings*. Cryptology ePrint Archive, Report 2019/099. 2019.

[Mic00] Silvio Micali. “Computationally Sound Proofs”. In: *SIAM Journal on Computing* 30.4 (2000). Preliminary version appeared in FOCS ’94., pp. 1253–1298.

[Pas03] Rafael Pass. “On Deniability in the Common Reference String and Random Oracle Model”. In: *Proceedings of the 23rd Annual International Cryptology Conference*. CRYPTO ’03. 2003, pp. 316–337.

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

[RU19] Barry Whitehat. *Rollup*. `https://github.com/barryWhiteHat/roll_up`. 2018.

[RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. “Interactive proofs of proximity: delegating computation in sublinear time”. In: *Proceedings of the 45th ACM Symposium on the Theory of Computing*. STOC ’13. 2013, pp. 793–802.

[SCI19] SCIPR Lab. *libiop: C++ library for IOP-based zkSNARKs*. 2019. URL: `https://github.com/scipr-lab/libiop`.

[SD19] StarkWare & 0x. *StarkDEX*. `https://www.starkdex.io/`. 2019.

[SN] Coda. *The SNARK Challenge*. `https://coinlist.co/build/coda`. 2019.

[Set19] Srinath Setty. *Spartan: Efficient and general-purpose zkSNARKs without trusted setup*. Cryptology ePrint Archive, Report 2019/550. 2019.

[Val08] Paul Valiant. “Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency”. In: *Proceedings of the 5th Theory of Cryptography Conference*. TCC ’08. 2008, pp. 1–18.

[XZZPS19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. “Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation”. In: *Proceedings of the 39th Annual International Cryptology Conference*. CRYPTO ’19. 2019, pp. 733–764.

[Zc14] Electric Coin Company. *Zcash Cryptocurrency*. `https://z.cash/`. 2014.

[bell15] Sean Bowe. *bellman: a zk-SNARK library*. 2015. URL: `https://github.com/zkcrypto/bellman`.

[mar19] SCIPR Lab. *A Rust library for the Marlin preprocessing zkSNARK*. 2019. URL: `https://github.com/scipr-lab/marlin`.