Fractal: Post-quantum and Transparent Recursive Proofs from Holography

Alessandro Chiesa(✉), Dev Ojha, and Nicholas Spooner(✉)

UC Berkeley, Berkeley, USA
{alexch,dojha,nick.spooner}@berkeley.edu

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. (The full version of this work is available at https://ia.cr/2019/1076.)

Keywords: Succinct arguments · Holographic proofs · Recursive proof composition · Post-quantum cryptography

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

A. Canteaut and Y. Ishai (Eds.): EUROCRYPT 2020, LNCS 12105, pp. 769–793, 2020.
https://doi.org/10.1007/978-3-030-45721-1_27

of the time of the prior computations. This *recursive composition* of SNARGs enables *incrementally verifiable computation* [56] and *proof-carrying data* [18,30]. 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 [18], 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 [18]. 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 [15], uses pairing-based SNARKs with a special algebraic property discussed below.[1] This has enabled real-world applications such as Coda [51], 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 [15] 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 [15] 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 [1,12,21,22]. Devising an efficient *transparent* (toxic-waste free) approach to recursion is an open problem.
- *It uses expensive algebra.* The approach of [15] 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

[1] A recent note sketches an alternative approach to recursion based on batch verification [23]. 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).

overheads. Attempts to find "better" cycles, without these limitations, have resulted in some negative results [26]. Indeed, finding *any other* cycles beyond MNT cycles is a challenging open problem.

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 Sect. 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 [7]. Recent work [27] has established a connection between holography and preprocessing (which we review in Sect. 1.2). The theorem below adds to this connection, by showing that interactive oracle proofs (IOPs) [14,52] that are holographic can be compiled into preprocessing SNARGs that are secure in the quantum random oracle model [20,28].

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.*

[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.

By applying Theorem 1 to known holographic proofs for non-deterministic computations (such as the PCP in [7] or the IPCP in [41]), 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 Sect. 1.2). 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*.

(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.)*

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 [13], and it offers succinct verification for all coefficient matrices compared to only structured ones as in [11].

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 kB to 200 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 Sect. 1.2), and are 2× bigger that the state of the art in post-quantum transparent *non*-preprocessing zkSNARKs [13]. 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 the full version [29] for performance details.)

Besides its application to post-quantum transparent recursion, our preprocessing zkSNARK provides attractive benefits over prior constructions, as we discuss in Sect. 1.2.

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) [18,30], 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 definition of PCD, see the full version [29].

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 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.*

[3] We stress that this step is a heuristic due to well-known limitations to the random oracle methodology [24,40]. Investigating how to provably instantiate the random oracle for many natural constructions is an active research frontier.

	R1CS instances	holographic?	indexer time	prover time	verifier time	round complexity	proof length	query complexity
[13]	arbitrary	NO	N/A	$O(m + n \log n)$	$O(\lvert\mathbb{x}\rvert + m)$	$O(\log n)$	$O(n)$	$O(\log n)$
[11] †	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	$O(m \log m)$	$O(m \log m)$	$O(\lvert\mathbb{x}\rvert + \log m)$	$O(\log m)$	$O(m)$	$O(\log m)$

Fig. 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 [11] reflect replacing the constant-query low-degree test in the construction with a concretely-efficient logarithmic-query low-degree test such as [9], to simplify comparison.

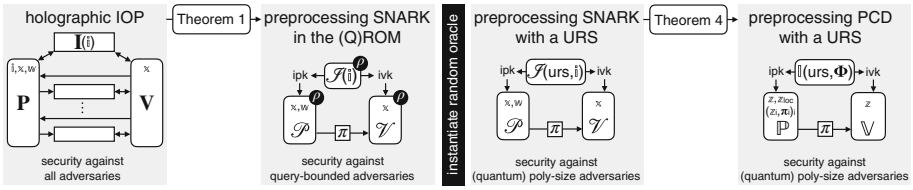

Fig. 2. Diagram of our methodology to recursive composition that is post-quantum and transparent.

1.2 Comparison with Prior Work

We provide a comparison with prior work in the three areas to which we contribute: holographic proofs (Sect. 1.2); preprocessing SNARGs (Sect. 1.2); and recursive composition of SNARKs (Sect. 1.2). 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., [10,57], and many others).

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 [7] 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 [17,33,35,46,53]). 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 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.

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 [7], 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 [41], 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 [7] and [41] 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 [27], 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 type	indexer time	prover time	verifier time
[7]	PCP	$\mathsf{poly}(N)$	$\mathsf{poly}(N)$	$\mathsf{poly}(\lvert\mathbb{x}\rvert + \log(N))$
[41]	IPCP	$\mathsf{poly}(N)$	$\mathsf{poly}(\lvert\mathbb{w}\rvert) + O(N)$	$O(\lvert\mathbb{x}\rvert + D \log W)$
this work	IOP	$O(N \log N)$	$O(N \log N)$	$O(\lvert\mathbb{x}\rvert + \log N)$

Fig. 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, [41] 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 [41] incorporate the techniques for linear-time sumcheck introduced in [57].)

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 [19,39,43,47], and have been studied in an influential line of work that has led to highly-efficient constructions (e.g., [44]) and large-scale deployments (e.g., [34]). 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 [45], and led to efficient constructions in [27,38,49]. 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 [27], 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 [8,55]).
  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 [37] 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 (weaker forms of preprocessing). Prior work proved recursive composition only for non-interactive arguments of knowledge with succinct verifiers [18]; 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, 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 [54]. 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 $O_\lambda(n^{1-1/c})$.

Recursion for pairing-based SNARKs. The approach to recursive composition of [15] uses pairing-based (preprocessing) SNARKs based on pairing-friendly cycles of elliptic curves. This approach applies to constructions with circuit-specific SRS (e.g. [44]) *and* to those with universal SRS (e.g. [27,38,45,49]).

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 [51] 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 [32]. 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 [26].

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 [44] 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 [9,16]. 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 Sect. 2.1 we explain how preprocessing simplifies recursive composition. In Sect. 2.2 we describe our compiler from holographic IOPs to preprocessing SNARGs (Theorem 1). In Sect. 2.3 we describe our efficient holographic IOP (Theorem 2), and then in Sect. 2.4 we discuss the corresponding preprocessing SNARG (Theorem 3). In Sect. 2.5 we describe how to obtain post-quantum and transparent PCD (Theorem 4). In Sect. 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. 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 "$\{z_i = F(z_{i-1})\text{"}\}_{i=1}^{T}$ and then inductively prove them. Informally, for $i = 1, \ldots, T$, 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 $\pi_i := \mathcal{P}(\mathsf{urs}, R, (R, i, z_i), t, \pi_i)$. 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 $\mathcal{V}(\mathsf{urs}, R, (R, i, z_i), t, \pi_i)$.[5]

We refer the reader to [18] 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

[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).

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 $\pi_i := \mathcal{P}(\mathsf{urs}, \mathsf{ipk}_R, (\mathsf{ivk}_R, i, z_i), \pi_i)$. 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 $\mathcal{V}(\mathsf{urs}, \mathsf{ivk}_R, (\mathsf{ivk}_R, i, z_i), \pi_i)$.

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.

In fact, preprocessing is *already* part of the efficient approach to recursive composition in [15]. 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 $\mathcal{R}$. For more details, see the full version [29].

Warmup: holographic PCPs. We first consider the case of PCPs, a special case of IOPs. Recall that the Micali transformation [50] 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 Sect. 2.3). Hence we are actually interested in compiling holographic IOPs. In this case our starting point is the BCS transformation [14], 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 [14]. 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 [27]. 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 [16]. 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 [13], 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 [48] 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 [13], 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 [11]. 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

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

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 [11]. 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 [13] 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. 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 [13]. (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 Sect. 2.2 (as captured in Theorem 1) to the efficient holographic proof for R1CS described in Sect. 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 the full version [29] 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 the full version [29] 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) [18,30], 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 Sect. 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 [18,56]) and for expressing dynamic distributed computations (such as MapReduce computations [31]).

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 [25,36]). 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 [24], 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.[7] The construction realizes recursive composition by following the template given

[6] The necessity to instantiate the random oracle before recursion also arises in the first construction of incrementally verifiable computation [56]. One way to circumvent this difficulty is to consider oracles that are equipped with a public verification procedure [30], 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).

in Sect. 2.1, except that the compliance predicate Φ may expect multiple input messages. This construction simplifies that of [18] 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.[8]

Security against quantum adversaries. A key feature of our result 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 [18] 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 Sect. 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 the full version [29] 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

[8] In contrast, the verification key ivk in [18] 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.

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 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* [2–4,6,42], 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 [4]). 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 [5,11,13] and for AIRs in [10].

We thus provide two constraint systems that target these two components. First, we provide a constraint system that realizes the FRI low-degree test [9], 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 [5,10,11,13].

Remark 2 (succinct languages). We stress that 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 [10] or Succinct-R1CS [11]. Doing so remains a challenging open problem, that would open up additional opportunities in recursive composition of *non*-preprocessing SNARKs.

References

1. Abdolmaleki, B., Baghery, K., Lipmaa, H., Siim, J., Zając, M.: UC-secure CRS generation for NARKs. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2019. LNCS, vol. 11627, pp. 99–117. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23696-0_6
2. Albrecht, M.R., et al.: Algebraic cryptanalysis of STARK-friendly designs: application to MARVELlous and MiMC. IACR Cryptology ePrint Archive, Report 2019/419 (2019)
3. Albrecht, M.R., et al.: Feistel structures for MPC, and more. IACR Cryptology ePrint Archive, Report 2019/397 (2019)
4. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. IACR Cryptology ePrint Archive, Report 2019/426 (2019)
5. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: lightweight sublinear arguments without a trusted setup. In: Proceedings of the 24th ACM Conference on Computer and Communications Security, CCS 2017, pp. 2087–2104 (2017)
6. Ashur, T., Dhooghe, S.: MARVELlous: a STARK-friendly family of cryptographic primitives. IACR Cryptology ePrint Archive, Report 2018/1098 (2018)
7. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, STOC 1991, pp. 21–32 (1991)
8. Barry Whitehat: Rollup (2018). https://github.com/barryWhiteHat/roll_up
9. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast Reed-Solomon interactive oracle proofs of proximity. In: Proceedings of the 45th International Colloquium on Automata, Languages and Programming, ICALP 2018, pp. 14:1–14:17 (2018)
10. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 701–732. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_23
11. Ben-Sasson, E., Chiesa, A., Goldberg, L., Gur, T., Riabzev, M., Spooner, N.: Linear-size constant-query IOPs for delegating computation. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11892, pp. 494–521. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36033-7_19

12. Ben-Sasson, E., Chiesa, A., Green, M., Tromer, E., Virza, M.: Secure sampling of public parameters for succinct zero knowledge proofs. In: Proceedings of the 36th IEEE Symposium on Security and Privacy, S&P 2015, pp. 287–304 (2015)
13. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_4. Full version available at https://eprint.iacr.org/2018/828
14. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_2
15. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 276–294. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_16. Extended version at http://eprint.iacr.org/2014/595
16. Ben-Sasson, E., Goldberg, L., Kopparty, S., Saraf, S.: DEEP-FRI: sampling outside the box improves soundness (2019). eCCC TR19-044
17. Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.P.: Robust PCPs of proximity, shorter PCPs, and applications to coding. SIAM J. Comput. **36**(4), 889–974 (2006)
18. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKs and proof-carrying data. In: Proceedings of the 45th ACM Symposium on the Theory of Computing, STOC 2013, pp. 111–120 (2013)
19. Bitansky, N., Chiesa, A., Ishai, Y., Paneth, O., Ostrovsky, R.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_18
20. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_3
21. Bowe, S., Gabizon, A., Green, M.: A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. Cryptology ePrint Archive, Report 2017/602 (2017)
22. Bowe, S., Gabizon, A., Miers, I.: Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050 (2017)
23. Bowe, S., Grigg, J., Hopwood, D.: Halo: recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021 (2019)
24. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM **51**(4), 557–594 (2004)
25. Chang, R., Chari, S., Ranjan, D., Rohatgi, P.: Relativization: a revisionistic retrospective. Bull. Eur. Assoc. Theor. Comput. Sci. **47**, 144–153 (1992)
26. Chiesa, A., Chua, L., Weidner, M.: On cycles of pairing-friendly elliptic curves. SIAM J. Appl. Algebra Geom. **3**(2), 175–192 (2019). https://arxiv.org/abs/1803.02067
27. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. Cryptology ePrint Archive, Report 2019/1047 (2019)

28. Chiesa, A., Manohar, P., Spooner, N.: Succinct arguments in the quantum random oracle model. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11892, pp. 1–29. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36033-7_1. Available as Cryptology ePrint Archive, Report 2019/834
29. Chiesa, A., Ojha, D., Spooner, N.: Fractal: post-quantum and transparent recursive proofs from holography (full version of this work). Cryptology ePrint Archive, Report 2019/1076 (2019). https://ia.cr/2019/1076
30. Chiesa, A., Tromer, E.: Proof-carrying data and hearsay arguments from signature cards. In: Proceedings of the 1st Symposium on Innovations in Computer Science, ICS 2010, pp. 310–331 (2010)
31. Chiesa, A., Tromer, E., Virza, M.: Cluster computing in zero knowledge. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 371–403. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_13
32. Coda: The SNARK Challenge (2019). https://coinlist.co/build/coda
33. Dinur, I., Reingold, O.: Assignment testers: towards a combinatorial proof of the PCP theorem. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2004, pp. 155–164 (2004)
34. Electric Coin Company: Zcash Cryptocurrency (2014). https://z.cash/
35. Ergün, F., Kumar, R., Rubinfeld, R.: Fast approximate probabilistically checkable proofs. Inf. Comput. **189**(2), 135–159 (2004)
36. Fortnow, L.: The role of relativization in complexity theory. Bull. Eur. Assoc. Theor. Comput. Sci. **52**, 229–244 (1994)
37. Freeman, D., Scott, M., Teske, E.: A taxonomy of pairing-friendly elliptic curves. J. Cryptol. **23**(2), 224–280 (2010). https://doi.org/10.1007/s00145-009-9048-z
38. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019)
39. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37
40. Goldwasser, S., Kalai, Y.T.: On the (in)security of the Fiat-Shamir paradigm. In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2003, pp. 102–113 (2003)
41. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. J. ACM **62**(4), 27:1–27:64 (2015)
42. Grassi, L., Kales, D., Khovratovich, D., Roy, A., Rechberger, C., Schofnegger, M.: Starkad and Poseidon: new hash functions for zero knowledge proof systems. IACR Cryptology ePrint Archive, Report 2019/458 (2019)
43. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_19
44. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
45. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10993, pp. 698–728. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_24

46. Gur, T., Rothblum, R.D.: Non-interactive proofs of proximity. In: Proceedings of the 6th Innovations in Theoretical Computer Science Conference, ITCS 2015, pp. 133–142 (2015)
47. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 169–189. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_10
48. Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM **39**(4), 859–868 (1992)
49. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: zero-knowledge SNARKs from linear-size universal and updateable structured reference strings. Cryptology ePrint Archive, Report 2019/099 (2019)
50. Micali, S.: Computationally sound proofs. SIAM J. Comput. **30**(4), 1253–1298 (2000). Preliminary version appeared in FOCS 1994
51. O(1) Labs: Coda Cryptocurrency (2017). https://codaprotocol.com/
52. Reingold, O., Rothblum, R., Rothblum, G.: Constant-round interactive proofs for delegating computation. In: Proceedings of the 48th ACM Symposium on the Theory of Computing, STOC 2016, pp. 49–62 (2016)
53. Rothblum, G.N., Vadhan, S.P., Wigderson, A.: Interactive proofs of proximity: delegating computation in sublinear time. In: Proceedings of the 45th ACM Symposium on the Theory of Computing, STOC 2013, pp. 793–802 (2013)
54. Setty, S.: Spartan: efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Report 2019/550 (2019)
55. StarkWare & 0x: StarkDEX (2019). https://www.starkdex.io/
56. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 1–18. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78524-8_1
57. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 733–764. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_24