How to Make Rational Arguments Practical and Extractable

Matteo Campanelli Chaya Ganesh, Rosario Gennaro

[1] Matter Labs
[2] Indian Institute of Science, Bangalore
[3] The City University of New York

Abstract. We investigate proof systems where security holds against *rational* parties instead of malicious ones. Our starting point is the notion of *rational arguments*, a variant of rational proofs (Azar and Micali, STOC 2012) where security holds against rational adversaries that are also *computationally bounded*.
Rational arguments are an interesting primitive because they generally allow for very efficient protocols, and in particular sublinear verification (i.e. where the Verifier does not have to read the entire input). In this paper we aim at narrowing the gap between literature on rational schemes and real world applications. Our contribution is two-fold.
We provide the first construction of rational arguments for the class of polynomial computations that is practical (i.e., it can be applied to real-world computations on reasonably common hardware) and with logarithmic communication. Techniques-wise, we obtain this result through a compiler from information-theoretic protocols and rational proofs for polynomial evaluation. The latter could be of independent interest.
As a second contribution, we propose a new notion of *extractability* for rational arguments. Through this notion we can obtain arguments where *knowledge* of a witness is incentivized (rather than incentivizing mere soundness). We show how our aforementioned compiler can also be applied to obtain efficient extractable rational arguments for NP.

Contents

1 Introduction

We study the problem of verifying that a statement is correct without having to re-execute the computation associated to it. In cryptographic literature a common primitive applied to this problem is a (cryptographic) *argument*. For deterministic (polynomial-time) computations, through an argument, we can convince a *weak* verifier, holding data D as input that $f(D) = y$ for some computation f. We can apply arguments to non-deterministic computations as well: a party holding a witness w can convince the verifier that it *knows* a witness such that $R(D, w) = 1$ for some NP relation R. This last type of arguments is also called an *argument of knowledge* (or *extractable* argument, since their "knowledge" is technically defined in terms of the existence of an extractor).

Arguments have become more and more practical. There is a long line of work spanning now more than a decade that has produced schemes that are extremely efficient in several dimensions (e.g. [BCG+13, WTs+18, CFQ19, MBKM19, CHM+20, CFF+21, ABC+22, LSTW21, KPV22] for a partial list). The most notable of their improvements is *fast verification*. They are able to achieve this by obtaining a proof size substantially smaller than the witness (resp. the execution trace) of the relation (resp. computation). At the same time they feature very performant proving algorithms. From now on we will refer to the arguments from this line of work as *very succinct arguments* for short.

Our goal: in this work we are interested in improving on very succinct arguments by keeping succinctness and the practicality of the prover and making the verifier *even faster*. In particular, we will aim for obtaining a *sublinear verifier* (the verifier does not even need to read the whole public input). Next, we shall motivate this problem and then describe the model in which we aim at achieving this.

Motivating sublinear verification.[1]

- *Motivation 1: overhead in verification from public input preprocessing.* In very succinct arguments, the efficiency of the verifier is often a consequence of how simple checking the proof is. Let us consider for example Groth16, currently one of the schemes with the "most minimalistic" proof and verifier [Gro16]. Consider a data string D (the public input) on which we want to verify property P. No matter the size of D or the complexity of the property P, a Groth16 proof consists of three group elements (A, B, C) and the verifier checks an equation of the form

$$e(A, B) \stackrel{?}{=} Y_D \cdot e(C, Z)$$

  where Z is a constant in the SRS (structured reference string) and Y_D is computed from the public input D. The verification check is as simple as it can get—at its core it involves two pairings only. However, for large public inputs, the bottleneck is from computing the element Y_D, which is defined as $\prod_i g_i^{D_i}$ for public generators g_i. For data as small as 2^{20} this computation can already be **three orders of magnitude** more expensive than the pairings themselves[2].

- *Motivation 2: lightweight access to data.* Processing a large public input may not be a bottleneck in all succinct arguments as it is in Groth16. Nonetheless, sublinear verification may still be advantageous in settings where it is impractical or strongly undesirable to access the whole public input. As a first example, a public input for a proof may be stored on the cloud, a fairly common practice for data in general. Downloading the whole input to verify a proof may be impractical when only a slow network is available (large parts of the world do not have access to fast internet). As another example, consider *light clients* in a blockchain. These are clients who do not intend to download the whole history of the chain but are still supposed to

[1]See also further discussion on sublinear verifier in Section 1.3.
[2]These estimates refer to the commonly used curve BLS12-381 [ECK+23].

retrieve information about it. In order to verify statements about the chain they could instead access only parts of it rather than the chain in its entirety.

We have so far motivated designing proof schemes with a sublinear verifier. Nevertheless, it is apparent that achieving such schemes within the *standard* model of security of interactive proofs/arguments is infeasible[3]. Sublinear verification has been achieved in only two models. One is that of *proofs of proximity* [RVW13] where the prover only certifies a close approximation to the exact value of the computation. One drawback of this model, is that efficiency of the protocol requires trading against correctness: this is no longer guaranteed to hold for instances that are only close to being in the language. For this reason and because we believe in the importance of modeling incentives explicitly, we will instead use the *rational* model of security described below.

Security against ~~malicious~~ rational provers. The standard notion of security in interactive proofs considers a malicious adversary who should not be able to convince the verifier of a false statement. In [AM12] Azar and Micali propose a variant of interactive proof where the prover is not considered honest or malicious but simply *rational*—i.e., choosing the strategy that will maximize their reward. In a *rational* security model, a verifier may *in principle* accept proofs of false statements. However, at the same time, a “rationally secure” scheme is designed so that it would be irrational for a prover to do so.

This model is applicable, for example, in settings where the prover has incentives to act honestly. These settings include cloud computing—where a server executes a computation and is going to be rewarded for its services—and blockchains—where it is easy to embed incentives in different layers of the system.

Protocols within the rational framework tend to have some intriguing features, including: simplicity, efficiency and the (already mentioned) possibility of a verifier which needs to query only a sublinear number of input positions. As an example, the protocol in [CG15] obtains a protocol for NC^1 with a verifier running in logarithmic time and querying a constant number of input bits; the interaction transcript has logarithmic size. The prover has essentially no overhead on top of performing the computation (it just needs to send certain sub-portions of the circuit evaluation).

From Proofs to Arguments. Not only rational proofs can give us light-weight verification protocols, but they may achieve all that without requiring any cryptographic assumption—the aforementioned protocol in [CG15] works even if it turns out that $\mathsf{P} = \mathsf{NP}$. But by introducing assumptions we can achieve even more: a line of work starting in [GHRV14] and [GHRV16] explores the additional *benefits of applying cryptographic assumptions* to verifiable computation *in a rational model* (this will be our focus). For example, in [GHRV14] Guo et al. show how to obtain two-message rational *arguments* (as opposed to *proofs*, which use no cryptographic assumption) for NC^1 with polylogarithmic verification. Later work has improved this result obtaining schemes with two messages for polynomial-time computations [GHRV16]. This last work uses relatively strong assumptions (subexponential FHE) and is arguably not concretely practical.

In this work we extend rational arguments to the realm of concrete practicality considering both deterministic and non-deterministic computations.

[3]Take the simple relation for parity where given a string $x \in \{0,1\}^n$ we say it has parity 1 iff $\sum_i x_i \equiv 1 \pmod 2$. Intuitively, we cannot convince a verifier of this without them reading the entire string. Why? Because by completeness of the proof system a string x_1 with parity 1 will have a high probability of being accepted by the verifier. But—and here is the absurd—so must be the accepting probability of an input x_0 identical to x_1 except in one position i where we flipped $x_1[i]$: since we are querying only a fraction of the input bits, the queries from x_0 and x_1 will be similarly distributed.

1.1 Our Contributions.

We continue the study of rational arguments. The scope of our contributions is *(i) practical*—we construct rational arguments for P more efficient than the state-of-the-art; *(ii) foundational*—we advance the theory of rational arguments by putting forth a new notion—*extractable* rational arguments—we build them for NP and argue their usefulness.
Rational Argument for P. We substantially improve the efficiency of rational arguments for polynomial computations. This work's starting point is the question:

Can we obtain rational arguments with (1) a sublinear verifier; and (2) whose prover is as efficient as some of the best available SNARKs?

We answer this question in the positive and construct the first rational argument for the class of polynomial computations that is practical (i.e., it can be applied to real-world computations on reasonably common hardware) and with logarithmic communication (see also Table 1). Our results provide improvements compared to both the *rational* and *cryptographic* arguments literature.

In comparison to prior rational arguments we obtain the first rational argument for deterministic computation that has a concretely efficient prover, has a logarithmic verifier using only public coin and is provably secure from simple assumptions (see Table 1). The only prior work on rational arguments for P ([GHRV16]) uses expensive primitive under the hood, required a polynomial blow-up of the prover time and had a polylogarithmic verifier. We note that our results require a higher number of messages: logarithmic instead of constant as in [GHRV16]. From a technical standpoint, we obtain our results from a very different blueprint. We show how to leverage the efficient modular constructions of recent SNARKs modeled as Algebraic Holographic Proofs[4] (or AHP [CHM+20]) and exploit techniques for rational arguments of polynomial evaluation, which may be of independent interest. The work in [GHRV16] instead closely follows the techniques from the delegation scheme of Kalai et al. [KRR14]: they construct δ-no-signaling *rational multi-prover proofs* (RMIPs). They then apply sub-exponentially secure Fully Homomorphic Encryption (FHE) scheme to transform no-signaling RMIPs into two-message rational arguments. Compared to other rational arguments for subclasses of P—e.g., the construction for NC circuits in [GHRV16]—ours removes the dependency of communication stemming from the depth of the circuit (logarithmic instead of polylogarithmic for a circuit of polylogarithmic depth).

In comparison to prior cryptographic arguments our constructions have several advantages. First, we are able to achieve a sublinear verifier. Second, our construction concretely improves on the efficiency of the prover by not requiring any cryptographic operations on what are usually called "computation commitments". A computation commitment is a commitment to a polynomial describing the computation. A significant amount of the prover's time is used to prove evaluations of these polynomials. This may require a significant number of cryptographic operations (usually consisting of several multi-scalar exponentiations in the size of the computations) and for some proof schemes this may be a source of efficiency bottleneck (see, e.g., Section 7.5 in [CGG+23]). See our instantiations for more details (Section 4.2). Finally, our constructions substantially simplify the "offline" stage (also called indexing) that is common in several cryptographic arguments (e.g., [CFF+21,CHM+20]). In this stage, the verifier is required to do a one-time preprocessing of the description of the computation. Intuitively, from the description of a computation C one can derive a key that can be to verify a proof on C efficiently. This process usually involves several multi-scalar exponentiations of a potentially large size (as large as the computation). This cost can be amortized over several proofs. However, it may still be infeasible for very weak devices wanting to verify large computations. Our construction allows verifier not to have to perform any cryptographic operations on the

[4]We note that our results could be easily adapted to the similar framework of PIOP [BFS20] and the more general framework of PHP [CFF+21].

Table 1: Comparison between this work and previous work on rational arguments for P. The parameter n is the instance size; T is the size of the computation. All asymptotic quantities implicitly contain a multiplicative factor polynomial in λ.

Scheme	Assumptions	Communication	P time	V time	Messages	Concretely efficient?	public coin?
[GHRV16]	subexp FHE	$O(\mathsf{polylog}(n))$	$\mathsf{poly}(T)$	$O(\mathsf{polylog}(n))$	2	✗	✗
This work	SXDH[5]	$O(\log(n))$	$O(T \log T)$	$O(\log(n))$	$O(\log(n))$	✓	✓

description of the computation. Instead it just needs to be able to have RAM (or network) access to its encoding and query a few of its points at verification time.

Rational Argument for NP. *We show that rational proofs that are succinct with noticeable reward gap*[6] *are impossible for* NP. This completes the landscape of results in succinct proofs for NP strengthening the impossibility results in [GHRV14] and [CG17].

The impossibility of succinct proofs motivates us to study succinct rational *arguments* for NP. *We put forth the notion of rational argument of knowledge, and construct a rational argument of knowledge for NP.* Our new notion models schemes where *knowledge* of a witness is incentivized (rather than incentivizing mere soundness). Our construction offers several advantages over succinct cryptographic arguments. First, the prover's cryptographic operations grows only with the size of the witness, and not with the input. This is useful for proving computations over *large public* data. Second, since our construction is essentially the same as our construction for P, we also get a verifier that is sublinear in the public data, and only has oracle access to it.

Rational Proof for Polynomial Evaluation. As a tool of independent interest, we construct a rational proof for polynomial evaluation, which we think of as a rational analog of a cryptographic polynomial commitment scheme. This allows outsourcing polynomial evaluation to the prover where the verifier only has oracle access to the description of the polynomial (either as a vector of coefficients or as tuple of point-evaluation pairs). Thus, this rational proof for polynomial evaluation is a rational analogue of a cryptographic polynomial commitment scheme, where the verifier is sublinear in the degree of the polynomial by virtue of having oracle access to the polynomial (and querying few places). This is in constrast to a verifier being sublinear in the degree of the polynomial since the cryptographic commitment being succinct in the degree. Indeed, we use this rational proof in our construction in lieu of a polynomial commitment scheme, for polynomials that encode public values (description of the function to be computed, public input).

1.2 Technical Overview

Recent constructions of zkSNARKs [CHM$^+$20, RZ21, CFF$^+$21, GWC19] follow a modular approach where an information-theoretic protocol is constructed in an abstract model like Probabilistically Checkable Proof (PCP), Interactive Oracle Proof (IOP) etc., and then the information-theoretic protocol is compiled into an argument system via a cryptographic compiler. Our construction is modular: starting from an information-theoretic protocol, our compiler uses cryptographic and rational proofs in order to compile into a rational argument.

AHP to SNARK. We use the formalization of Algebraic Holographic Proofs (AHP) from [CHM$^+$20]. In an AHP the prover receives as inputs a statement x and a witness w. In each round of interaction with the verifier, it sends oracle polynomials and the verifier responds with a random challenge. Then, in a query phase, the verifier queries the oracle polynomials at evaluation point. For oracle p and evaluation query z, it obtains $v = p(z)$.

[5] This assumption is used when we instantiate our construction with the Dory polynomial commitment [Lee21]. See also Section 4.2. For a formal definition of SXDH, see Definition 1 in [Lee21].

[6] The reward gap bounds the difference between the reward of the honest and dishonest provers.

Finally, based on the result of these evaluation queries the verifier outputs a bit indicating "accept" or "reject". An AHP can be turned into an argument system by replacing the oracles and the query phase with a polynomial commitment scheme (PCS) [KZG10]. In the argument, the prover commits to the polynomials obtained from the information-theoretic prover, and then upon receiving an evaluation point z, the prover uses the evaluation proof of the PCS to to convince the verifier that the claimed evaluation v is indeed $v = p(z)$ given a commitment to p.

Our main idea. Typically, the oracles sent by an AHP prover consist of the following: witness-carrying polynomials (WCP) (polynomials that encode the witness vector w) and index polynomials or computation commitments (which encode the description of the computation or the index i). The AHP verifier encodes the public input x into a polynomial and makes queries to the WCP and index polynomials. When compiled into an argument, the prover performs cryptographic work for the PCS to commit to WCP and index polynomials. Our first idea is to move the index polynomials from being cryptographically committed with a PCS to being available to the verifier as an "oracle" and *providing* rational proofs about evaluations. While typically, oracle access to the verifier is an "intermediate" model and oracles have to be realized in the real-world, our oracle model of index polynomials/computation commitments is limited to processing the index and storing a certain encoding in memory. This is part of the verifier's preprocessing (which is anyway done in cryptographic arguments too), but our oracle modeling is to capture the property that the verifier *does not read the entire preprocessed material.* Cryptographic arguments achieve sublinearity of the verifier in the index i by preprocessing the computation into a short cryptographic commitment, with respect to which the prover gives evaluation proofs of the PCS. We achieve sublinearity of the verifier in the index i by preprocessing the computation into an encoding; the verifier only queries this encoding at a few points. This makes the verifier sublinear in the size of the computation, but we also need to achieve sublinearity in the public input x. Here too, the verifier has oracle access to x and queries it a few points during verification. Our idea to achieve sublinearity in x is as follows. First, we observe that the computation that makes the verifier linear in x in the underlying AHP is in evaluating a polynomial encoding of x at a random point. We outsource this computation to the prover, so the prover provides this evaluation together with a rational proof of correct evaluation. Crucially, this rational proof can be verified given just oracle access to x. We note that x need not be preprocessed in any way, it is assumed to be stored in the cloud and the verifier accesses it by querying at certain positions. A key technical tool towards our construction is a rational proof for polynomial evaluation which we describe next.

Rational proofs for polynomial evaluation. Given a computation described as an arithmetic circuit of size n and depth d, the protocol of [CG15] works as follows. On input x, the prover sends the claimed output y to the verifier, and then they engage in a recursive protocol. The prover sends the two input value y_L, y_R to the output gate g, the verifier checks that $g(y_L, y_R) = y$, and then chooses one of y_L and y_R at random to recurse on. For uniform circuits, the verifier complexity is $O(d)$. We then describe polynomial evaluation as a parallel circuit that has depth that is logarithmic in the degree. Now, invoking the above protocol on this circuit, we obtain a rational proof with $O(\log d)$ verification where the verifier only needs oracle access to the polynomial, for example as a vector of d coefficients.

Defining rational arguments with extractable properties. In classical arguments of knowledge we require that "if the verifier accepts a proof with a reasonable probability, then we are able to (efficiently) extract a valid witness by interacting with the prover". Translating this notion into the rational setting requires care.

First, it does not seem possible to have a meaningful notion for instances not in the language (this is not the problem in the standard notion). Here is why: Recall that the goal of designing a rational protocol is to *incentivize* parties towards the “right” choice, but for an input $x \notin L$ there is no way of incentivizing them in such a manner. This is because if $x \notin L$ then the only rational strategy of any prover is to just run the protocol (earning a possibly meager reward) even if it doesn’t know the witness (a witness the prover *cannot* possibly know). For this reason a meaningful definition can only make guarantees about instances $x \in L$[7], e.g., discrete logarithm.

Also, the notion should link the capability of the extractor of outputting a witness to the reward of the prover. Intuitively, we would like to guarantee that only a prover who knows the witness should obtain the highest reward. This notion is hard to capture directly. Our definition tries to instead capture this intuition:

A prover not knowing the witness is incentivized to search for it and only then run the protocol.

When is it reasonable to ask for the prover to search for a witness? Whenever the witness is *moderately* hard to find or hard to find for the verifier (but not for the prover). For example, maybe there is a (quasi-polynomial time or high-degree polynomial time) search algorithm that a prover with lots of computational power could run. Or, the witness could be moderately hard to find in a non-complexity theoretic sense (it may require for example downloading and scanning the whole history of a blockchain or the Wikipedia corpus). Naturally, this is meaningful if the cost of the resources required for such provers to search for the witness is compensated by the received reward. See Section 1.3 and Appendix C for further discussion. We provide more details and intuitions on our definition for extractable rational arguments in Section 5.1.

Our final constructions. Our construction works as follows. We look at the polynomial oracles sent by the AHP prover that “encode” the witness, called the witness-carrying polynomials (WCP), and polynomial oracles arising from the indexer (preprocessing), that encode the computation (index). The prover in the compiled argument will treat the encodings of index and witness differently. The argumenting indexer runs the AHP indexer and outputs these index polynomials as oracles for the verifier. Then the argument prover commits to the WCPs sent by the AHP prover using a polynomial commitment scheme (PCS) and sends this commitment to the argument verifier. Now, *(i)* queries to WCP are answered by giving a PCS evaluation proof; *(ii)* queries to index polynomials are answered by giving a rational proof; *(iii)* the verifier’s linear step in the AHP decision algorithm is outsourced to the prover who provides a rational proof of correct evaluation of the encoding of the input. This construction (which is essentially a compiler) works for both P and NP. For NP, this also satisfies our notion of proof of knowledge. We extract WCPs from the knowledge soundness of the PCS and decode the witness from it. We show that this is a valid witness with respect to the oracle index and oracle input by relying on the guarantees of the rational proof. Assuming that PCS is knowledge sound, if extraction fails, then the prover gave an inconsistent answer in the rational proof. Now, if the rational proof for polynomial evaluation has a noticeable reward gap, then, we show that the probability of extraction failure affects the reward gap of the final prover.

1.3 Discussion: Alternative Sublinear Verifiers; Applications

On other approaches for large public inputs

There exist other techniques to let the verifier not read the whole public input in the context of arguments. They can be used as a way to achieve a sublinear verifier. These approaches, however, have some drawbacks which justify the alternative framework proposed in this paper. One folklore approach, for example, is to let the verifier preprocess the public input

[7]The cryptographic literature has precedents for definitions with “one-sided” guarantees, i.e. for either yes or no instances only. Such an example is semantic security in witness encryption [GGSW13].

D and keep only its digest $h = H(D)$, where H is some collision-resistant hash function that is fast to compute, such as SHA256. Now, instead of providing a proof for just the statement "$f(D) = y$" the prover will certify the augmented statement "$f(D) = y \wedge H(D) = h$".

One problem with this approach is that it substantially increases the proving time because hash functions commonly tend to be expensive when used proof systems (see, e.g., discussion in [WYX+21]). This problem can be mitigated using *Hash-and-Prove* schemes [FFG+16] but this requires strong assumptions on the hash function. The latter also needs to be "algebraic", hence orders-of-magnitude slower to compute for the verifier.

Another limitation of the digest-based approach arises in settings where the data are dynamic. If the data change, then the digest h will change as well. Also, depending on the hash function used, the digest will plausibly need to be recomputed from scratch after each update. Rational arguments can arguably provide a simpler approach since they do not require processing again the input after an update.

Application scenarios for the rational model

Applications for rational arguments for deterministic computations:

- **Outsourced computations (cloud and volunteer computing):** In cloud computing businesses buy computing time from a service, rather than maintain their own computing resources . In volunteer computing systems (such as SETI@Home or Folding@Home [KWA+01, P+10]), where the rewards are non-fungible "points". Rational arguments are suitable to both of these settings since the computing parties have an incentive through the payment (resp. reward points) they earn.
- **Smart contract execution:** Smart contracts are programs running on a blockchain[8]. Since it is costly to run them on chain, a common desideratum on contracts is for them to be as simple as possible. The sublinearity of the verification algorithm in rational arguments is an attractive feature because it may allow a smart contract not to even read its whole input to perform an action. As an example, consider a contract that releases some funds to a user if they provide y such that $y = f(x)$ where x is some value stored in the contract. The contract could be, e.g., enforcing the "incentives for an outsourced computation f" as outlined above or representing a transition function of its own state through f (in this case the reward is tantamount to some form of mining). A party could now provide y to the contract together with a proof π[9]. The contract would then query the required positions of x (a sublinear amount) and then release the funds depending on the reward function of the rational argument.

Applications for rational arguments of knowledge: in general, rational arguments of knowledge are a natural primitive for the following problem: party A is interested in rewarding someone (party B) if they know data with a specific feature. Party A does not necessarily need to see the data, it just need to be persuaded that they are "extractable" from someone. Some examples are: a foundation willing to reward someone for finding a proof of the Riemann Hypothesis, or private individuals willing to reward anyone storing specific parts of Wikipedia or the Internet Archive on a decentralized network for storage (see Section 7.2 in [CFK22] for more discussion of these and other examples).

We also provide one more application for extractable rational arguments that is specific to blockchains, in particular for the problem of *data availability*. In blockchains like Ethereum not all nodes have access to the full data of the chain. Nodes that are not expected to store the full data, *light nodes*, may verify that others do store them by

[8]See, e.g., `https://ethereum.org/en/developers/docs/smart-contracts`.

[9]Here we assume a non-interactive rational arguments. Our arguments are interactive, but public-coin. We leave it as an open problem to which extent rational arguments can be made non-interactive through Fiat-Shamir.

downloading very small random chunks of such data (data availability sampling). The more assurance we want to have that someone is storing those data, the more light nodes need to query overall. Rational arguments of knowledge may provide a different approach to the problem where full nodes are provided incentives for storing the full data through a rational argument for the statement "I know (specific positions) of data D that is the opening of this public Merkle Tree root".

Understanding the advantages the solutions above in the applications we mentioned (as well others) may require a game-theoretic analysis of the specific setting, which is out of the scope of this paper. In Appendix C, however, we provide some general observations for when this approach may be viable, its limitations and heuristic mitigations.

1.4 Related Work

Interactive Proofs and Arguments. *Interactive proofs* allow a powerful prover to convince a computationally bounded verifier of the correctness of a computational statement via the power of interaction and randomness. These proofs allow delegation of computation where a computationally weak client to delegate computation tasks to a powerful server in a verifiable way – the server returns the result of the computation together with a proof, and the client can then verify that the output returned by the server is indeed correct while performing work less than what is necessary for computing the function itself. *Soundness* property of the proof system guarantees that no matter what the prover does, it cannot make the verifier accept a false claim. Many applications also require *succinct verification*, where the verifier is able to check a nondeterministic polynomial-time computation in time that is much shorter than the time required to run the computation given a the NP witness. In interactive *arguments*, soundness is guaranteed only against provers that are computationally bounded, and argument systems allow us to circumvent efficiency shortcomings of proofs. Interactive succinct arguments were constructed by Kilian [Kil92], and was made non-interactive by Micali [Mic94] soon after, in the random oracle model.
Delegation of Computation—Proofs for P. Real world problems common in the

delegation of compuration scenario often correspond to complexity classes lower than NP. The work of Goldwasser, Kalai and Rothblum [GKR08] gave a single-round argument to verifiably delegate any bounded depth computation where the verifier is quasi-linear. The work of Kalai, Raz and Rothblum [KRR14] achieves a single-round argument with quasi-linear verification time for any language in P. The latter result is under standard assumptions, in contrast to non-standard assumptions that are known to be necessary for single round succinct arguments for NP.

Rational Proofs and Arguments. Rational proofs, introduced by Azar and Micali [AM13] are a framework of interactive proofs where the prover is relaxed to be rational rather than malicious. In a rational proof, the prover is incentivized: the verifier pays the prover according to the quality of the result provided, and this reward is set up so that it is irrational for the prover to return the result of the computation incorrectly. Azar and Micali also illustrated the power of rational proofs by constructing a single-round rational proof for all of #P. The work of Campanelli and Gennaro [CG15, CG17] constructs rational proofs with composition properties (e.g., that are reusable for multiple executions) for bounded-depth circuits and bounded-space computations. The work of Guo, Hubacek, Rosen and Vald [GHRV14] restrict the rational prover to be computationally bounded, obtaining the notion of *rational arguments.* The advantage of rational arguments over their classical counterparts is that they allow for low communication and a *sublinear* verifier. [GHRV14] construct sublinear verifier rational arguments for class NC^1. The work of [GHRV16] extends this result by constructing a single-round rational argument with sublinear verification for class P.

Another line of work achieves verifiable computation against rational parties in an indirect manner through approaches based on *fine-grained* cryptographic primitives [CG18].

2 Preliminaries

Notation. We denote a finite field by $\mathbb{F}$. We denote by λ a security parameter. We consider interactive arguments for relations, where a prover P convinces the verifier that it knows a witness w such that for a public statement $x, (x, w) \in \mathcal{R}$. For a pair of PPT interactive algorithms P, V, we denote a protocol by (P, V) and by $(P(w), V)(x)$, the random variable representing the transcript between P and V when interacting where w is (an optional) P's private input and x is a common input. We denote by $\mathsf{out}((P(w), V)(x))$ the output of V after interacting with P on input x. We denote a *negligible* function, a function that vanishes faster than the inverse of any polynomial in λ by $\mathsf{negl}(\lambda)$. A *noticeable* function is the inverse of a polynomial.

2.1 Modelling Access to Inputs

Given algorithm A, we write $A(x)$ (as it is standard) to denote that the string x is written on the input tape of A. This implies in particular that A will have to run at least in time $|x|$. Some of the algorithms in this paper will require only oracle access to inputs. We denote them through the following superscript notation as in the following:

$$A^{[u]}(x)$$

Above A has oracle access to input u but "standard" (tape) access to input x. Algorithms may have oracle and standard access to more than one input like in the following example:

$$A^{[u],[v]}(x, y)$$

where the algorithm A has oracle access to inputs u, v and standard access to inputs x, y.

2.2 Rational Proofs and Arguments

We give the definition of Rational Proofs from [AM12]. Let $\mathsf{rew}(\cdot)$ denote a randomized function computed by V on the transcript. The goal of a rational P is to maximize the expected value of reward rew, while the goal of V is to learn the correct evaluation of the function f on x.

Definition 1 (Rational Proof)**.** A circuit $F : \{0,1\}^n \to \{0,1\}^*$ admits a rational proof if there exists an interactive proof (P, V) and a randomized reward function $\mathsf{rew} : \{0,1\}^* \to \mathbb{R}_{\geq 0}$ such that

1. For any input $x \in \{0,1\}^n$, $\Pr[\mathsf{out}((P(x), V^{[x]})) = F(x)] \geq 1 - \mathsf{negl}(n)$.
2. For every prover $\widetilde{P}$, and for any input $x \in \{0,1\}^n$ there exists a $\delta_{\widetilde{P}}(x) \geq 0$ such that $\mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))] + \delta_{\widetilde{P}}(x) \leq \mathbb{E}[\mathsf{rew}((P(x), V^{[x]}))]$.

The expectations and probabilities are taken over the coins of both prover and verifier.

Rational arguments introduced in [GHRV14] capture a rational prover that is restricted to computationally bounded strategies.

Definition 2 (Rational Argument [GHRV14])**.** A circuit $F : \{0,1\}^n \to \{0,1\}^*$ admits a rational argument if there exists an interactive protocol (P, V) and a randomized reward function $\mathsf{rew} : \{0,1\}^* \to \mathbb{R}_{\geq 0}$ such that for any input $x \in \{0,1\}^n$, any prover $\widetilde{P}$ of size $\leq 2^{\lambda(n)}$:

1. $\Pr[\mathsf{out}((P,V)(x)) = F(x)] \geq 1 - \mathsf{negl}(\lambda)$
2. $\mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))] \leq \mathbb{E}[\mathsf{rew}((P(x), V^{[x]}))] + \mathsf{negl}(\lambda)$
3. If $\Pr[\mathsf{out}((P,V)(x)) \neq F(x)] \geq 1/p(n)$ for some polynomial $p(\cdot)$, then there exists a polynomial $q(\cdot)$ such that $\mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))] + 1/q(n) \leq \mathbb{E}[\mathsf{rew}((P(x), V^{[x]}))]$

The expectations and the probabilities are taken over the random coins of the prover and verifier. The rational argument is efficient if the verifier runs in time $o(n)$.

Property 1 corresponds to the notion of completeness, Property 2 guarantees that the gain attained by deviating from the prescribed strategy in a computationally bounded way is at most negligible. Property 3 guarantees that not reporting the correct output with noticeable probability results in a noticeable loss in reward.

Reward gap measures how big the loss of a prover that always reports $F(x)$ incorrectly is. A noticeable gap in expectation between such a prover and the prescribed behavior guarantees that it is beneficial for the prover to act honestly to significantly increase its utility, allowing us to argue for rationality in the presence of computational cost.

Definition 3 (Reward Gap [GHRV14])**.** Let (P, V) be a rational argument for a function $f : \{0,1\}^n \to \{0,1\}^*$ with reward function rew. Let $\epsilon_{\widetilde{P}} = \Pr[\mathsf{out}((\widetilde{P}, V)(x)) \neq F(x)]$, and $\delta_{P^*}(x) = \mathbb{E}[\mathsf{rew}((P(x), V^{[x]}))] - \mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))]$. The reward gap is a function $\Delta : \mathbb{N} \to \mathbb{R}$ such that for every $n \in \mathbb{N}$,

$$\Delta(n) = \min_{x} \min_{P^* : \epsilon_{P^*} = 1} [\delta_{P^*}(x)]$$

For an arbitrary prover $\widetilde{P}$ we have $\delta_{\widetilde{P}}(x) \geq \epsilon_{\widetilde{P}} \cdot \Delta(x)$.

Examples of Rational Proofs. For concreteness here we show the protocol for a single threshold gate (readers are referred to [AM12, AM13, GHRV14] for more examples). Let $G_{n,k}(x_1, \ldots, x_n)$ be a threshold gate with n Boolean inputs, that evaluates to 1 if at least k of the input bits are 1. The protocol in [AM13] to evaluate this gate goes as follows. The Prover announces the number $\tilde{m}$ of input bits equal to 1, which allows the Verifier to compute $G_{n,k}(x_1, \ldots, x_n)$. The Verifier select a random index $i \in [1..n]$ and looks at input bit $b = x_i$ and rewards the Prover using Brier's Rule $BSR(\tilde{p}, b)$ where $\tilde{p} = \tilde{m}/n$ i.e. the probability claimed by the Prover that a randomly selected input bit be 1. Then

$$BSR(\tilde{p}, 1) = 2\tilde{p} - \tilde{p}^2 - (1 - \tilde{p})^2 + 1 = 2\tilde{p}(2 - \tilde{p})$$

$$BSR(\tilde{p}, 0) = 2(1 - \tilde{p}) - \tilde{p}^2 - (1 - \tilde{p})^2 + 1 = 2(1 - \tilde{p}^2)$$

Let m be the true number of input bits equal to 1, and $p = m/n$ the corresponding probability, then the expected reward of the Prover is

$$pBSR(\tilde{p}, 1) + (1 - p)BSR(\tilde{p}, 0) \tag{1}$$

which is easily seen to be maximized for $p = \tilde{p}$ i.e. when the Prover announces the correct result. Moreover one can see that when the Prover announces a wrong $\tilde{m}$ his reward goes down by $2(p - \tilde{p})^2 \geq 2/n^2$. In other words for all n-bit input x, we have $\Delta(x) = 2/n^2$ and if a dishonest Prover $\widetilde{P}$ cheats with probability $\epsilon_{\widetilde{P}}$ then $\delta_{\widetilde{P}} > 2\epsilon_{\widetilde{P}}/n^2$.

2.3 Interactive Proofs

Indexed Relations. In order to achieve a succinct verifier, we model relations as triples instead of pairs; this allows us to split the verifier's input into one part for the offline phase and one part for the online phase. The offline input, called the index, is encoded by an indexer algorithm, and this encoding can be reused for proofs over different instances for the same index. Additionally, the *encoded* index allows verifier efficiency – in cryptographic protocols the encoding is a cryptographic digest and reading it is more effcieint than reading the index; in rational protocols, the encoding once generated and stored, is accessed as an oracle at very few locations.

Definition 4 (Indexed relation [CHM+20]). An *indexed relation* $\mathcal{R}$ is given by a set of triples $(\mathsf{i}, \mathsf{x}, \mathsf{w})$ where i is the index, x is the instance, and w is the witness. The corresponding *indexed language* $\mathcal{L}_{\mathcal{R}}$ is the set of pairs (i, x) for which there exists a witness w such that $(\mathsf{i}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$.

Rank-1 Constraint Systems (R1CS). *Rank-1 Constrained Systems (R1CS)* are a popular way to encode computations to be proven via a SNARK. R1CS were implicitly defined as *Quadratic Arithmetic Programs* in [GGPR13] where it is proven that they are NP-complete (and therefore can express any arbitrary non-deterministic polynomial computation).

An R1CS instance is a tuple $(\mathbb{F}, A, B, C, x, N, m)$ where x denotes the public input of the instance, $A, B, C \in \mathbb{F}^{N\times N}$ are matrices defined over a field $\mathbb{F}$, with $N \geq |x| + 1$, and there are at most m non-zero entries in each matrix.

An R1CS instance $(\mathbb{F}, A, B, C, x, N, m)$ is satisfiable if there exists a witness $w \in \mathbb{F}^{N-|x|-1}$ such that

$$(A \cdot z) \circ (B \cdot z) = (C \cdot z)$$

where $z = (x, 1, w)$, $\cdot$ is the matrix-vector product, and $\circ$ is the Hadamard product.

R1CS for deterministic computations. The above Indexed relation definition is for NP, can be used to capture P as well. A tuple $(\mathsf{i}, \mathsf{x}, \mathsf{w})$ corresponds to $(f, (x, y), z)$ where $f(x) = y$ and z consists of all intermediate values in computing $f(x)$. Now, R1CS representation for this tuple is well-defined.

A proof/argument system for a language $\mathcal{L}$ allows a prover P to convince a verifier V that $x \in L$ for a common input x. A proof of knowledge intuitively captures not only the truth of a statement $x \in L$, but also that the prover is in "possession" of a witness w.

2.4 Polynomial Commitment Scheme

A polynomial commitment scheme [KZG10] allows a prover to open evaluations of the committed polynomial succinctly. A polynomial commitment scheme over $\mathbb{F}$ is given by a tuple $\mathsf{PC} = (\mathsf{setup}, \mathsf{commit}, \mathsf{open}, \mathsf{eval})$ where:

- $\mathsf{setup}(1^\lambda, D) \rightarrow \mathsf{pp}_{\mathsf{pc}}$. On input security parameter λ, and an upper bound $D \in \mathbb{N}$ on the degree, setup generates public parameters $\mathsf{pp}_{\mathsf{pc}}$.
- $\mathsf{commit}(\mathsf{pp}_{\mathsf{pc}}, f(X), d) \rightarrow (C, \tilde{\mathbf{c}})$. On input the public parameters $\mathsf{pp}_{\mathsf{pc}}$, and a univariate polynomial $f(X) \in \mathbb{F}[X]$ with degree at most $d \leq D$, commit outputs a commitment to the polynomial C, and additionally an opening hint $\tilde{\mathbf{c}}$.
- $\mathsf{open}(\mathsf{pp}_{\mathsf{pc}}, f(X), d, C, \tilde{\mathbf{c}}) \rightarrow b$. On input the public parameters $\mathsf{pp}_{\mathsf{pc}}$, the commitment C and the opening hint $\tilde{\mathbf{c}}$, a polynomial $f(X)$ of degree $d \leq D$, open outputs a bit indicating accept or reject.
- $\mathsf{eval}(\mathsf{pp}_{\mathsf{pc}}, C, d, x, v; f(X)) \rightarrow b$. A public coin interactive protocol $\langle P_{\mathsf{eval}}(f(X)), V_{\mathsf{eval}}\rangle(\mathsf{pp}_{\mathsf{pc}}, C, d, z, v)$ between a PPT prover and a PPT verifier. The parties have as common input public parameters $\mathsf{pp}_{\mathsf{pc}}$, commitment C, degree d, evaluation point x, and claimed evaluation v. The prover has, in addition, the opening $f(X)$ of C, with $\deg(f) \leq d$. At the end of the protocol, the verifier outputs 1 indicating accepting the proof that $f(x) = v$, or outputs 0 indicating rejection.

A polynomial commitment scheme must satisfy completeness, commitment binding, evaluation binding. It may also satisfy stronger properties like extractability and hiding.

Definition 5 (Completeness)**.** For all polynomials $f(X) \in \mathbb{F}[X]$ of degree $d \le D$, for all $x \in \mathbb{F}$,

$$\Pr\left(b = 1 \ : \ \begin{array}{c} \mathsf{pp_{pc}} \leftarrow \mathsf{setup}(1^\lambda, D) \\ (C, \tilde{\mathbf{c}}) \leftarrow \mathsf{commit}(\mathsf{pp_{pc}}, f(X), d) \\ v \leftarrow f(x) \\ b \leftarrow \mathsf{eval}(\mathsf{pp_{pc}}, C, d, x, v; f(X)) \end{array} \right) = 1.$$

Definition 6 (Commitment Binding)**.** A polynomial commitment scheme PC is binding if for all PPT $\mathcal{A}$, the following probability is negligible in λ:

$$\Pr\left(\begin{array}{c} \mathsf{open}(\mathsf{pp_{pc}}, f_0, d, C, \tilde{\mathbf{c}}_\mathbf{0}) = 1 \wedge \\ \mathsf{open}(\mathsf{pp_{pc}}, f_1, d, C, \tilde{\mathbf{c}}_\mathbf{1}) = 1 \wedge \\ f_0 \neq f_1 \end{array} \ : \ \begin{array}{c} \mathsf{pp_{pc}} \leftarrow \mathsf{setup}(1^\lambda, D) \\ (C, f_0, f_1, \tilde{\mathbf{c}}_\mathbf{0}, \tilde{\mathbf{c}}_\mathbf{1}, d) \leftarrow \mathcal{A}(\mathsf{pp_{pc}}) \end{array} \right).$$

Definition 7 (Extractability)**.** For any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a PPT algorithm $\mathcal{E}$ such that the following probability is negligible in λ:

$$\Pr\left(b = 1 \wedge \mathcal{R}_{\mathsf{eval}}(\mathsf{pp_{pc}}, C, x, v; \tilde{f}, \tilde{\mathbf{c}}) = 0 \ : \ \begin{array}{c} \mathsf{pp_{pc}} \leftarrow \mathsf{setup}(1^\lambda, D) \\ (C, d, x, v, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp_{pc}}) \\ (\tilde{f}, \tilde{\mathbf{c}}) \leftarrow \mathcal{E}^{\mathcal{A}_2}(\mathsf{pp_{pc}}) \\ b \leftarrow \langle \mathcal{A}_2(\mathsf{st}), V_{\mathsf{eval}} \rangle (\mathsf{pp_{pc}}, C, d, x, v) \end{array} \right).$$

where the relation $\mathcal{R}_{\mathsf{eval}}$ is defined as follows:

$$\mathcal{R}_{\mathsf{eval}} = \{ \big((\mathsf{pp_{pc}}, C \in \mathbb{G}, \ x \in \mathbb{F}, \ v \in \mathbb{F}); \ (f(X), \tilde{\mathbf{c}}) \big) : (\mathsf{open}(\mathsf{pp_{pc}}, f, d, C, \tilde{\mathbf{c}}) = 1) \wedge v = f(x) \}$$

Definition 8 (Succinctness)**.** The scheme is *proof succinct* if the commitments and the evaluation proofs are of size independent of the degree of the polynomial, $|C|$ is $\mathsf{poly}(\lambda)$, $|\pi|$ is $\mathsf{poly}(\lambda)$ where π is the transcript obtained by applying FS to eval. Additionally, the scheme is *verifier succinct* if eval runs in time $\mathsf{poly}(\lambda) \cdot \mathsf{log}(d)$ for the verifier.

2.5 Algebraic Holographic Proof

Definition 9 (AHP [CHM+20])**.** An *Algebraic Holographic Proof (AHP)* over a field family $\mathcal{F}$ for an indexed relation $\mathcal{R}$ is given by the following tuple:

$$\mathsf{AHP} = (\mathsf{k}, \mathsf{s}, \mathsf{d}, \mathcal{I}, \mathcal{P}, \mathcal{V})$$

where $\mathsf{k}, \mathsf{s}, \mathsf{d} : \{0,1\}^* \to \mathbb{N}$ are polynomial-time computable functions; $\mathcal{I}, \mathcal{P}, \mathcal{V}$ are the *indexer*, *prover*, and *verifier* algorithms; k denotes the number of rounds, s denotes the number of polynomials in each round, and d specifies degree bounds on these polynomials. The protocol proceeds as follows:

- **Indexing phase** The indexer $\mathcal{I}$ receives as input a field $\mathbb{F} \in \mathcal{F}$, index i for $\mathcal{R}$, and outputs $\mathsf{s}(0)$ polynomials $p_{0,1}, \ldots, p_{0,s(0)} \in \mathbb{F}[X]$ of degrees at most $\mathsf{d}(|\mathsf{i}|, 0, 1), \ldots, \mathsf{d}(|\mathsf{i}|, 0, \mathsf{s}(0))$ respectively. This phase does not depend on the public input or witness and simply consists of encoding the given index i.
- **Online phase** The prover $\mathcal{P}$ receives[10] $(\mathsf{i}, \mathsf{x}, \mathsf{w})$, for an instance x and witness w such that $(\mathsf{i}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$. The verifier $\mathcal{V}$ receives x and oracle access to the polynomials output by $\mathcal{I}(\mathbb{F}, \mathsf{i})$. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ interact over $\mathsf{k} = \mathsf{k}(|\mathsf{i}|)$ rounds. In the i-th round, $i \in [\mathsf{k}]$, the verifier $\mathcal{V}$ sends a message $\rho_i \in \mathbb{F}^*$ to the prover $\mathcal{P}$; the prover $\mathcal{P}$ responds with $\mathsf{s}(i)$ oracle polynomials $p_{i,1}, \ldots, p_{i,s(i)} \in \mathbb{F}[X]$. At the end of k rounds, the verifier outputs additional randomness $\rho_{\mathsf{k}+1} \in \mathbb{F}^*$ which is an auxiliary input to $\mathcal{V}$ in subsequent phases.

[10] $\mathbb{F}$ is an implicit input to all algorithms, and omitted for brevity.

- **Query phase** Let $\mathbf{p} = (p_{i,j})_{i\in[\mathsf{k}],j\in[\mathsf{s}(i)]}$ be a vector consisting of all the polynomials sent by the prover $\mathcal{P}$. The verifier $\mathcal{V}$ executes a subroutine $\mathsf{Q}_{\mathcal{V}}$ that receives $(\mathsf{x}; \rho_1, \ldots, \rho_{\mathsf{k}+1})$ and outputs a query set Q consisting of tuples $((i, j), z)$ that are interpreted as "query $p_{i,j}$ at $z \in \mathbb{F}$". We denote a vector consisting of query answers by $\mathbf{p}(Q)$.
- **Decision phase** The verifier outputs accept or reject based on the answers received to the queries and its randomness. That is, $\mathcal{V}$ executes a subroutine $\mathsf{D}_{\mathcal{V}}$ that receives $(\mathsf{x}, \mathbf{p}(Q); \rho_1, \ldots, \rho_{\mathsf{k}+1})$ as input, and outputs a decision bit.

The function d determines what kind of provers are considered for the completeness and soundness properties of the proof system. A (potentially malicious) prover $\tilde{\mathcal{P}}$ is considered *admissible* for AHP if, in an interaction with the verifier $\mathcal{V}$, it holds that for every round $i \in [\mathsf{k}]$ and oracle index $j \in [s(i)]$ we have $\deg(p_{i,j}) \leq \mathsf{d}(|\mathsf{i}|, i, j)$. The honest prover $\mathcal{P}$ is required to be admissible under this definition. An AHP satisfies completeness and soundness as defined below.

- Completeness: An AHP is complete if for all $\mathbb{F} \in \mathcal{F}$ and any $(\mathsf{i}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$, the decision bit returned by $\mathcal{V}^{\mathcal{I}(\mathbb{F},\mathsf{i})}(\mathsf{x})$ after interacting with an honest $\mathcal{P}(\mathsf{i}, \mathsf{x}, \mathsf{w})$ is 1.
- Soundness: An AHP is ϵ-sound if for any prover $\mathcal{P}^*$, field $\mathbb{F} \in \mathcal{F}$, for every $(\mathsf{i}, \mathsf{x}) \notin \mathcal{L}_{\mathcal{R}}$, and auxiliary input z:

$$\Pr[\langle \mathcal{P}^*(\mathsf{i}, \mathsf{x}, z), \mathcal{V}^{\mathcal{I}(\mathbb{F},\mathsf{i})}(\mathsf{x})\rangle = 1] \leq \epsilon$$

Using the compilation process in [BFS20, CHM+20], $\mathsf{AHP} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ for $\mathcal{R}$ can be turned into a *preprocessing argument*, denoted by $\mathsf{AoK} = (\mathsf{S}, \mathsf{I}, \mathsf{P}, \mathsf{V})$ for $\mathcal{R}$.

3 Definitions and Building Blocks

3.1 Rational Arguments with Indexing

We now formalize preprocessing the function so that a rational verifier can be sublinear in the size of the function representation (for instance, sublinear in the size of the circuit computing f). Once the preprocessing is done, the verifier has oracle access to preprocessed information and can then engage in multiple proofs for the same function by using the same oracles. We call this model rational argument with *indexing* to indicate that the function is preprocessed or indexed and the verifier only needs this indexed material in order to compute the reward. Crucially, accessing this indexed material is via oracle queries, and the verifier only makes sublinear number of queries.

Definition 10 (Rational Argument with Indexing)**.** A Rational Argument with Indexing for a function $f : \{0,1\}^n \to \{0,1\}^*$ is a tuple of four algorithms $(\mathsf{Setup}, \mathsf{idx}, \mathsf{Prv}, V, \mathsf{rew})$. Setup is a probabilistic polynomial-time setup algorithm that samples public parameters $\mathsf{pp}_{\mathsf{rat}}$. The indexer algorithm idx is a deterministic algorithm that takes $\mathsf{pp}_{\mathsf{rat}}, f$, and outputs $\phi \in \{0,1\}^*$ that is used as an oracle by V. $(\mathsf{Prv}, V^{[\phi]}, \mathsf{rew})$ is a rational argument for f where the reward function $\mathsf{rew}^{[\phi]}(\mathsf{pp}_{\mathsf{rat}}, x, (\mathsf{Prv}, V)(x)) \to \mathbb{R}_{\geq 0}$ can be computed with oracle access to ϕ.

In our constructions, the verifier needs to do no preprocessing for a function being indexed except saving a representation of the function f. In particular, when the function is polynomial evaluation, the verifier just needs to store the polynomial (for example, as a vector of coefficients) and make few oracle accesses to it during verification.

A note on input access. The standard verifier in interactive proofs needs to read the whole public input (i.e., the input is written on an input tape and read sequentially by a Turing machine). Works on rational proofs [AM12] model the verifier's access to input differently. As in proofs of proximity [RVW13], we do not require the verifier to read all of it, but only to query specific indices (usually randomly sampled). In this work, we exploit this fact plus more: we explicitly model the fact that the verifier may also have oracle access to (some description of) the function and oracle access to the input. This provides efficiency of the verifier when combined with the fact that, in the online stage, the verifier needs to query very few points in the representation of f. $(\mathsf{Prv}, V^{[\phi],[x]}, \mathsf{rew})$ is a rational argument for f with *both function and input indexing* where the reward function $\mathsf{rew}^{[\phi],[x]}(\mathsf{pp}_{\mathsf{rat}}, x, (\mathsf{Prv}, V)(x)) \to \mathbb{R}_{\geq 0}$ can be computed with oracle access to ϕ and x.[11]

Definition 11 (Rational Argument with Full Indexing)**.** A Rational Argument with Full Indexing for a function $f : \{0,1\}^n \to \{0,1\}^*$ is a tuple of four algorithms $(\mathsf{Setup}, \mathsf{idx}, \mathsf{Prv}, V, \mathsf{rew})$. Setup is a probabilistic polynomial-time setup algorithm that samples public parameters $\mathsf{pp}_{\mathsf{rat}}$. The indexer algorithm idx is a deterministic algorithm that takes $\mathsf{pp}_{\mathsf{rat}}, f$, and outputs $\phi \in \{0,1\}^*$ that is used as an oracle by V. $(\mathsf{Prv}, V^{[\phi],[x]}, \mathsf{rew})$ is a rational argument for f where the reward function $\mathsf{rew}^{[\phi],[x]}(\mathsf{pp}_{\mathsf{rat}}, x, (\mathsf{Prv}, V)(x)) \to \mathbb{R}_{\geq 0}$ can be computed with oracle access to ϕ and x.

3.2 Efficient Rational Proofs for Polynomial Evaluation

Let $f \in \mathbb{F}[X]$ be a univariate polynomial of degree d, $f(X) = \sum_{i=0}^{d} a_i X^i$. We now consider rational proofs for polynomial evaluation. That is for the function

$$F_{\text{poly},f}(t) := f(t)$$

In order to construct *efficient* rational proofs for polynomial evaluation we exploit parallel circuits as much as possible proceed as follows:

- We describe a highly parallel circuit for $F_{\text{poly},f}$ (see $\mathsf{PolyEvalPreproc}$ in Fig. 1) which works by requiring a small preprocessing by the verifier. This preprocessing requires logarithmic steps in the degree of the polynomial. Its output is described as the vector $\mathbf{u}$ in Fig. 2.
- We then apply a rational proof for highly parallel arithmetic circuits from [CG15] (see also Lemma 2) to $\mathsf{PolyEvalPreproc}$.

Theorem 1. *There exists a rational proof for polynomial evaluation where:*

- *the verifier runs in time $O(\log(d))$;*
- *the query and the round complexity is $O(\log(d))$;*
- *the prover runs in time $O(d)$;*
- *the reward gap is noticeable.*

Proof. Completeness and rationality properties of the protocol follow by applying Lemma 2 and Lemma 1 to the protocol in Fig. 2.

The efficiency of the protocol follows from these observations: In order to evaluate f at point t, the sum consists of $d+1$ terms where each term can be computed using $O(d)$ multiplications in the circuits after a preprocessing. This results into an arithmetic circuit of depth $O(\log d)$ as described in Fig. 1. We can thus obtain a rational proof that runs in logarithmic rounds and communication and with a logarithmic verifier (Theorem 1). □

In the previous theorem we used the following lemma, which we prove in Appendix E.1.

Lemma 1. *The following properties hold for the circuit* $\mathsf{PolyEvalPreproc}$ *in Fig. 1:*

[11] See also Section 2.1.

```
PolyEvalPreproc$(f, \mathbf{u})$

Parse $\mathbf{u}$ as $(u_1, \ldots, u_\ell)$
Parse $f$ as a tuple of coefficients s.t. $f(X) = \sum_{i=0}^{2^\ell} a_i X^i$, $\mathsf{deg}(f) = 2^\ell$
for $i = 1, \ldots, 2^\ell$ (in parallel)
    $y_i \leftarrow \prod_{j \in [\ell] : b_j(i) = 1} u_j$   (where $b_j(i)$ is the $j$-th bit of $i$ for each $j$)
    $s_i \leftarrow a_i \cdot y_i$
endfor
$z \leftarrow a_0 + \sum_{i \in [2^\ell]} s_i$
return $z$
```

Figure 1: An arithmetic circuit with logarithmic depth for polynomial evaluation.

Rational proof for $F_{\text{poly},f}(t)$:

- Setup:
  - Do nothing
- idx(f):
  - return a description of f as a tuple of coefficients
- Protocol $P(f, t) \leftrightarrow V^{[f]}(t)$:
  - P and V compute $\mathbf{u} = (t, t^2, t^4, \ldots, t^{2^{\ell-1}}, t^{2^\ell})$ by repeated squaring where $\mathsf{deg}(f) = 2^\ell$
  - The two parties then proceed by running RPPar.$P_F \leftrightarrow$ RPPar.$V_F^{[x]}$ where the circuit $F :=$ PolyEvalPreproc and input $x := (f, \mathbf{u})$.
- Reward:
  - Return reward from RPPar.rew$^{[x]}$

Figure 2: Rational proof for polynomial evaluation. The protocol RPPar is from the statement of Lemma 2.

- *if f is a polynomial and $t \in \mathbb{F}$ then* PolyEvalPreproc$\left(f, \left(t, t^2, t^4, \ldots, t^{2^{\ell-1}}, t^{2^\ell}\right)\right)$ *outputs $f(t)$;*
- *the circuit wiring can be decided in time logarithmic in the degree of the polynomial.*

A description of the protocol RPPar used in Fig. 2 is in the appendix (Appendix B).

Lemma 2 ([CG15]). *Let $\mathbb{F}$ be a field and F be a logtime-uniform arithmetic circuit of logarithmic depth then there exists a rational proof* RPPar *for F. For an input x of size n, the verifier runs in logarithmic time in n, the query and the round complexity is logarithmic in n. Its reward gap is noticeable.*

4 Our Compiler: AHP to Rational Argument for P

Consider the indexed relation $\mathcal{R}$ given by triples $(\mathsf{i}, \mathsf{x}, \mathsf{w})$ that represent $(f, (x, y), w)$ such that $f(x) = y$ and w consists of all intermediate values in computing $f(x)$. We now construct a compiler that transforms an AHP into a rational argument for deterministic computations.

The high-level idea behind our compiler is along the lines of the AHP to succinct argument compiler [BFS20, CHM+20]. In an AHP, the prover and the verifier interact where the prover sends polynomial oracles, and the verifier sends random challenges. Then, the verifier queries the polynomials at some challenge points for evaluation, and finally accepts or rejects. Existing compilers use a polynomial commitment scheme as a cryptographic object to realize the polynomial oracle. The prover in the compiled argument commits to the polynomials and then provides evaluations together with proofs of correct evaluation. The argument verifier accepts if both the PCS verifier and the AHP verifier accept. Our key idea is to separate the queries, and have the prover answer some of them with a cryptographic proof and the rest with a rational proof.

- Queries to index polynomials $p_{0,j}$: on query z, prover responds with y and a rational proof for the statement "$p_{0,j}(z) = y$".
- Queries to witness polynomials: $p_{i,j}$: on query z, prover responds with y together with an evaluation proof of a polynomial commitment scheme for "$p_{i,j}(z) = y$ AND $C_{i,j} = \mathsf{Commit}(p_{i,j})$"

Our compiler removes cryptographic operations in the indexing step and the prover performs cryptographic operations that grows only with the input and not the circuit. The resulting argument is a rational argument. Now, in order to make the verifier sublinear in the public input, we outsource the computation involving the public input to the prover, and have the prover provide a rational proof.

Additional properties of AHP. We use the definition of witness-carrying polynomials (WCPs) [ABC+22] that identify a minimum set of polynomials containing enough information about the whole witness, with which auxiliary commitments are shown to be consistent. We also restrict the AHP extractor to be straightline and deterministic so that it is essentially a witness decoding algorithm that works for both honest and malicious provers. Additionally, we assume that the public input access in the computation of the AHP verifier is restricted to querying a polynomial encoding of it. We note that the AHPs of PLONK, Marlin and Sonic already satisfy these properties in Appendix F.

4.1 A Formal Description of our Compiler from AHP to Rational Argument

Let $\mathsf{AHP} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ be an AHP for $\mathcal{R}$. Let $\mathsf{PC} = (\mathsf{setup}, \mathsf{commit}, \mathsf{open}, \mathsf{eval})$ be a polynomial commitment scheme. and $\mathsf{rat\text{-}pc} = (\mathsf{idx}, P \leftrightarrow V, \mathsf{rew})$ be a rational argument system for polynomial evaluations. We assume that $\mathcal{V}$ has a query algorithm $\mathsf{Q}_{\mathcal{V}}$ and a decision algorithm $\mathsf{D}_{\mathcal{V}}$. Furthermore, whenever $\mathsf{D}_{\mathcal{V}}$ uses the statement x, it is for the computation of a polynomial encoding $\hat{\mathsf{x}}$ and therefore giving oracle access to $\hat{\mathsf{x}}$ suffices to correctly run the decision algorithm $\mathsf{D}_{\mathcal{V}}$. We therefore think of $\mathsf{D}_{\mathcal{V}}(\mathsf{x}, \mathbf{v}; \rho_1, \ldots, \rho_{\mathsf{k}+1})$ as $\mathsf{D}^{\hat{\mathsf{x}}}_{\mathcal{V}}(\mathbf{v}; \rho_1, \ldots, \rho_{\mathsf{k}+1})$. This is without loss of generality for AHPs of interest (Appendix F). The rational argument with full indexing $(\mathsf{Setup}, \mathsf{idx}, \mathsf{P}, \mathsf{V}, \mathsf{rew})$ for $\mathcal{R}$ is described in Fig. 3.

Theorem 2. *Let $\mathcal{R}$ be the relation* $(\mathsf{i}, \mathsf{x}, \mathsf{w})$ *that represents* $(f, (x, y), w)$ *Let* AHP *be an AHP for $\mathcal{R}$ with soundness error ϵ_1, let* PC *be a polynomial commitment scheme with extractability error ϵ_2, and* $\mathsf{rat\text{-}pc}$ *be a rational proof for polynomial evaluation with reward gap Δ. The, the compiled system is a rational argument for $\mathcal{R}$ with reward gap $\Delta(1-\epsilon_1-\epsilon_2)$. The verifier runs in time $O(|y| + \log |f| + \log |x|)$ given oracle access to x, where $|f|$ is the R1CS complexity of f.*

[12]Here, we assume that the AHP encodes the input as a polynomial in monomial basis. The compiler extends to AHPs that encode input as a polynomial in Lagrange basis as well, by simply using a rational proof for evaluating a polynomial given point-evaluation pairs. We elaborate in Appendix F.

Let $\mathsf{AHP} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ be an AHP for $\mathcal{R}$. Let $\mathsf{PC} = (\mathsf{setup}, \mathsf{commit}, \mathsf{open}, \mathsf{eval})$ be a polynomial commitment scheme. Let $\mathsf{rat\text{-}pc} = (\mathsf{idx}, P \leftrightarrow V, \mathsf{rew})$ be a rational argument system for polynomial evaluations.

- Setup $\mathsf{Setup}(1^\lambda)$: Run $\mathsf{pp_{pc}} \leftarrow \mathsf{setup}(1^\lambda), \mathsf{pp_{rat}} \leftarrow \mathsf{Setup}(N)$. Output $\mathsf{srs} := (\mathsf{pp_{pc}}, \mathsf{pp_{rat}})$.
- Preprocessing idx: On input i:
  1. Run the AHP indexer $\mathcal{I}$ on i to obtain polynomials $(p_{0,j})_{j=1}^{\mathsf{s}(0)} \in \mathbb{F}[X]$.
  2. For $j \in [\mathsf{s}(0)]$, invoke the rational argument indexer to preprocess $p_{0,j}$; $\phi_j \leftarrow \mathsf{idx}(p_{0,j})$.
  3. Input indexing: Run the rational argument indexer idx on x; $\phi_v \leftarrow \mathsf{idx}(\mathsf{x})$.

  Output $\mathsf{ipk} := (\phi_v, (\phi_j)_{j=1}^{\mathsf{s}(0)})$.
- Prover P and Verifier V: In every round $i \in [\mathsf{k}]$, P and V simulate the interaction between the AHP prover $\mathcal{P}(\mathsf{i}, \mathsf{x}, \mathsf{w})$ and verifier $\mathcal{V}(\mathsf{x})$. In round i:
  1. V interacts with P and internally runs $\mathcal{V}$. V receives $\rho_i \in \mathbb{F}$ from $\mathcal{V}$, and forwards it to P.
  2. P interacts with V and internally runs $\mathcal{P}$. P forwards the received ρ_i to $\mathcal{P}$, and receives $\mathsf{s}(i)$ polynomials $p_{i,1}, \ldots, p_{i,\mathsf{s}(i)} \in \mathbb{F}[X]$. P invokes the polynomial commitment scheme to commit to each of these polynomials.
  $$C_{i,j} = \mathsf{commit}(p_{i,j}), \text{ for } j = 1, \ldots, \mathsf{s}(i)$$
  P sends the vector of commitments $\mathbf{C} = \{C_{i,j}\}$ to V.
- P and V simulate the query phase of the AHP.
  1. Let $\mathbf{p} = (p_{i,j})_{i \in [\mathsf{k}], j \in [\mathsf{s}(i)]}$ denote the vector consisting of all the polynomials sent by $\mathcal{P}$, and $\mathbf{C}$ the vector of commitments to $\mathbf{p}$. $\mathcal{V}$ executes $\mathsf{Q}_{\mathcal{V}}^{\mathsf{x}}(\rho_1, \ldots, \rho_{\mathsf{k}+1})$ and outputs a query set Q consisting of tuples $(p_{i,j}, z)$. V separates the set of query points $(p_{i,j}, z)$ into rational queries Q_r (that are to index polynomials, that is, $i = 0$), and crytographic queries Q_c ($i \neq 0$) and forwards (Q_c, Q_r) to P.
  2.
     - *Rational queries* ($i = 0$)*:* For $(p_{0,j}, z_j) \in Q_r$, P and V run a rational argument for polynomial evaluation. For each $(f_j, z_j) \in Q_r$, execute rational proof for $F_{\text{poly},f_j}(z_j)$: $P(f_j, z_j) \leftrightarrow V^{[\phi_j]}(z_j)$.
     - *Cryptographic queries* ($i > 0$)*:* For $(p_{i,j}, z) \in Q_c$, P and V run $\mathsf{eval}(\mathsf{pp_{pc}}, \mathbf{C}, d, \mathbf{z}, \mathbf{v}; \mathbf{p})$ where $\mathbf{z}$ is the vector of all query points in Q_c and $\mathbf{v}$ is vector of all claimed evaluations. In the above, eval is the *batched* evaluation protocol of the polynomial commitment scheme that proves the evaluation of multiple polynomial commitments.
- Reward phase:
  1. V runs $\mathcal{V}$'s decision algorithm $\mathsf{D}_{\mathcal{V}}^{\hat{\mathsf{x}}}(\mathbf{v}; \rho_1, \ldots, \rho_{\mathsf{k}+1})$. Whenever $\mathsf{D}_{\mathcal{V}}$ queries $\hat{\mathsf{x}}$ on α, P and V run a rational argument for polynomial evaluation. For each $(\hat{\mathsf{x}}, \alpha_j)$, execute rational proof for $F_{\text{poly},\hat{\mathsf{x}}}(\alpha_j)$: $P(\hat{\mathsf{x}}, \alpha_j) \leftrightarrow V^{[\hat{\mathsf{x}}]}(\alpha_j)$[12].
  V receives a decision bit b_1 from $\mathsf{D}_{\mathcal{V}}$.
  2. Let b_2 be the decision bit of V from the execution of eval.
  3. If $(b_1 = 1 \wedge b_2 = 1)$, for each $(p_{0,j}, z_j) \in Q_r$, V computes $\mathsf{rew}_j \leftarrow \mathsf{rew}^{[\phi_j]}(\mathsf{pp_{rat}}, z_j, (P, V)(z_j))$, and sets $\mathsf{rew} \leftarrow \min_j\{\mathsf{rew}_j\}$.

Figure 3: Our compiler from AHP to rational arguments.

Proof. Let $\mathsf{AHP}^{\{p_{i,j}\}}$ be the starting AHP with polynomial oracles. We first consider a hybrid intermediate oracle argument system $\mathsf{AHP}^{\{p_{0,j}\}}$ where the polynomial oracles of $\{p_{i,j}\}$, for $i \neq 0$ are realized via a cyptographic PCS PC, and the polynomials $\{p_{0,j}\}$ are still available as oracles. Suppose prover $\tilde{\mathsf{P}}$ of $\mathsf{AHP}^{\{p_{0,j}\}}$ convinces V of instance $(f, x, \tilde{y})$ in $L_{\mathcal{R}}$. We construct an adversary $\mathcal{A}_{\mathsf{pc}}$ against the extractability game of PC. $\mathcal{A}_{\mathsf{PC}}$ receives the commitment key ck, and random coins and internally invokes $\tilde{\mathsf{P}}$ to obtain a set of commitments $\{C_{i,j}\}_{i\in[k],j\in[s(i)]}$. We now invoke the extractor $\mathcal{E}_{\mathsf{PC}}$, which given the same input as $\mathcal{A}_{\mathsf{PC}}$ outputs a set of polynomials $\tilde{p} = \{p_{i,j}\}_{i\in[k],j\in[s(i)]}$. If $\mathcal{E}_{\mathsf{pc}}$ fails, that is the extracted polynomials are inconsistent with the alleged evaluations ($\tilde{p}(\mathbf{z}) \neq \mathbf{v}$), then $\mathcal{A}_{\mathsf{PC}}$ wins the extractability game, which happens with probability ϵ_2. Hence, except with probability ϵ_2, $\tilde{p}(\mathbf{z}) = \mathbf{v}$. Given that the evaluations are valid w.r.t the set of polynomials $\tilde{p}$, V accepts when AHP verifier accepts. Since the soundness error of the AHP is ϵ_1, the probability that V accepts $(f, x, \tilde{y}) \notin L_{\mathcal{R}}$ is $\epsilon_1 + \epsilon_2$.

Now, we replace oracle calls to $\{p_{0,j}\}$ polynomials with rational proofs for polynomial evaluation. The hybrid protocol $\mathsf{AHP}^{\{p_{0,j}\}}$ is compiled into a rational argument Π where $p_{0,j}$ and $\hat{\mathsf{x}}$ are realized via a rational proof system. To show the reward gap of Π, we consider the following events:

- Event A: V accepts, that is, $b_1 = 1 \wedge b_2 = 1$, and the rational prover correctly answers all queries to $p_{0,j}), \hat{\mathsf{x}}$, that is, for all j, $\mathsf{out}((P^*, V)(z)) = p_{0,j}(z)$, for every query z to $p_{0,j}$, and $\mathsf{out}((P^*, V)(z)) = \hat{\mathsf{x}}(z)$, for every query z to $\hat{\mathsf{x}}$.
- Event B: V accepts, that is, $b_1 = 1 \wedge b_2 = 1$, and the rational prover incorrectly answers queries, that is, for some j, $\mathsf{out}((P^*, V)(z)) \neq p_{0,j}(z)$.
- Event C: V rejects, that is $b_1 = 0 \vee b_2 = 0$

Now, the expected reward is

$$\mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))] = \Pr[A] \cdot \mathbb{E}[\mathsf{rew}((\widetilde{P}, V)(x_A))|A] + \Pr[B] \cdot \mathbb{E}[\mathsf{rew}((\widetilde{P}, V)(x_B))|B] + \Pr[C] \cdot \mathbb{E}[\mathsf{rew}((\widetilde{P}, V)(x_C))|C] \quad (2)$$

where x_E denotes the distribution of the instance for the rational argument given the occurance of event E. When event C occurs, by construction, the reward is 0, therefore,

$$\mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))] = \Pr[A] \cdot \mathbb{E}[\mathsf{rew}((\widetilde{P}, V)(x_A))|A] + \Pr[B] \cdot \mathbb{E}[\mathsf{rew}((\widetilde{P}, V)(x_B))|B]$$

Now, we bound the probability of event A.

$$\Pr[A] \leq \epsilon_1 + \epsilon_2$$

This is because in the hybrid world where only rational proofs are oracles and others are cryptographically realized, the soundness error is $\epsilon_1 + \epsilon_2$, and when event A occurs, the prover gives correct answers to all ratuonal queries (and therefore behave ideally). Moreover, the prover in the compiled argument does not learn anything more from the query itself since in a public-coin AHP, the queries are independent of the rest of the communication. Now, we have

$$\mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))] \leq \Pr[A] \cdot \mathbb{E}[\mathsf{rew}((\widetilde{P}, V)(x_A))|A] + (1 - \Pr[A]) \cdot (\mathbb{E}[\mathsf{rew}((P, V)(x_B))] - \Delta)$$

In a public-coin AHP, the verifier queries to the statement polynomials are from tossing public coins, the distribution of queries to polynomials of rational proof is independent of the transcript and instance. Thus, x_A and x_B are distributed identically. Therefore,

$$\mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))] \leq \Pr[A] \cdot \mathbb{E}[\mathsf{rew}((P, V)(x))] + (1 - \Pr[A]) \cdot (\mathbb{E}[\mathsf{rew}((P, V)(x))] - \Delta)$$

$$\mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))] \leq \mathbb{E}[\mathsf{rew}((P, V)(x))] - (1 - \Pr[A]) \cdot \Delta$$

$$\mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))] \leq \mathbb{E}[\mathsf{rew}((P, V)(x))] - (1 - \epsilon_1 - \epsilon_2) \cdot \Delta$$

□

4.2 Instantiating Our Compiler

Instantiating our compiler requires selecting an AHP (with a decision algorithm with specific properties), a polynomial commitment scheme and a rational argument system for polynomial evaluation. The last can be instantiated through the construction described in Section 3.2. Below we discuss AHP and polynomial commitment candidates. These instantiations are also suitable for our construction in Section 5.2.

We show how to make the decision algorithm $\mathsf{D}_{\mathcal{V}}$ sublinear in the statement x by realizing $\mathsf{D}_{\mathcal{V}}^{\mathsf{x}}(\mathbf{v}; \rho_1, \ldots, \rho_{\mathsf{k}+1})$ for candidate AHPs: Marlin in Appendix F.2, PLONK in Appendix F.3, and Sonic in Appendix F.4.

For polynomial commitments, example candidate constructions are Dory [Lee21] (provably extractable under the SXDH assumption, with transparent setup, logarithmic communication/rounds/verification) and KZG [KZG10] (non-interactive, with trusted setup, constant communication and verification but with extraction provable in the AGM [FKL18]).

5 Rational Proofs and Arguments of Knowledge for NP

The work of [GHRV14] showed how to downscale rational proofs for #P of Azar and Micali to NP. They give a succinct (what they call extremely laconic) rational proof for any language in NP where the communication is one bit. However, there are two downsides: (i) the verifier is not succinct, and (ii) the reward gap is negligible. We begin by exploring the possibility of succinct rational proofs for NP with noticeable reward gap.

Succinct Rational Proofs for NP. [GHRV14] also shows that such succinct rational proofs for NP cannot have non-negligible reward gap when the proof is public coin and non-interactive. We show that we cannot have rational proofs for NP that have communication complexity logarithmic in the size of the witness, even when interaction is allowed.[13]
We prove the following theorem in Appendix E.2.

Theorem 3 (Impossibility of *succinct* rational proofs for NP)**.** *Let (P, V) be a public coin interactive rational proof for a language L with perfect completeness. Let* rew *be the reward function,* $\log(n)$ *and* $\mathsf{poly}(n)$ *be the communication complexity and verification complexity respectively. If the reward gap Δ is a noticeable function, then L is decidable* $\mathsf{poly}(n)$ *time.*

This theorem shows that even in the rational setting (as in the classical setting), we cannot have succinctness by increasing the number of rounds if we want a *proof* system. We therefore relax the soundness requirement from proof to *argument* just as in the classical setting to achieve succinctness. Most languages that show up in practice need the guarantee of knowledge of a witness: for instance proof of discrete logarithm of a public value or preimage of a compressing function do not guarantee much since they always exist. Instead, what is meaningful is if the prover actually knows the discrete logarithm or the preimage. In the classical setting, this is captured by proofs/arguments of knowledge. We now define *rational arguments of knowledge* for NP.

5.1 Succinct Rational Arguments of Knowledge for NP

We now define rational arguments of knowledge where probability of the extractor outputting a witness is tied to the reward of the prover (see also discussion in Section 1.2).

[13]Theorem 16 of [GHRV14] shows that rational proofs with *one round* and logarithmic communication for NP are impossible, assuming NP $\not\subseteq$ BPP. The theorem we show extends the impossibility even for rational proofs with logarithmically many rounds. Theorem 5 of [CG17] shows that rational proofs with polylog rounds and communication complexity for NP are impossible, assuming NP $\not\subseteq$ BPQP. Our theorem is under the weaker assumption that NP $\not\subseteq$ BPP.

Intuitions on our notion

Recall from Section 1.2 that our notion is silent on the case $x \notin L$. An informal version of our definition is as follows:

For every $x \in L$:

1. the honest prover (running the protocol with some valid witness as input) will earn the highest reward;
2. a prover not knowing the witness should earn a "low" amount.

The two requirements together capture the intuition discussed in Section 1.2, i.e. that a prover not knowing a witness will be incentivized to search for it before protocol execution:

- Requirement 2 implies that not knowing the witness (as in "acting in a way that makes the extractor fail") gives a low reward.
- By requirement 1, there is at least one strategy that provides the high reward (the honest prover's strategy).
- Note that requirement 1 and 2 together prevent the extractor from being trivial (never returning valid witnesses for statements in the language).

The two requirements are formalized in Definition 12. See further discussion in Appendix C.

Definition 12 (Rational AoK)**.** A Rational AoK Π for $\mathcal{L} \in NP$ is given by a tuple $(\mathsf{Setup}, \mathsf{idx}, P, V, \mathsf{rew})$: Setup is a probabilistic polynomial-time setup algorithm that samples public parameters $\mathsf{pp}_{\mathsf{rat}}$. The indexer algorithm idx is a deterministic algorithm that takes $\mathsf{pp}_{\mathsf{rat}}, \mathcal{R}$, and outputs $\phi \in \{0,1\}^*$ that is used as an oracle by V, where $\mathcal{R}(x,\cdot) = 1$ iff $x \in \mathcal{L}$. The reward function $\mathsf{rew}^{[\phi],[x]}(\mathsf{pp}_{\mathsf{rat}}, x, (P,V)(x)) \to \mathbb{R}_{\geq 0}$ can be computed with oracle access to ϕ and x. Π is said to have reward gap Δ if for any PPT prover $\tilde{P}$, any input $x \in \mathcal{L}$, the following hold:

1. (Completeness) $\mathsf{rew}(P,V)(x) = O(1)$[14]
2. (Knowledge soundness) There exists a PPT extractor $\mathcal{E}$ such that if $\Pr[w \leftarrow \mathcal{E}^{\tilde{P}}(x) : \mathcal{R}(x,w) = 0] \geq 1/q(n)$ for some polynomial $q(\cdot)$, then there exists a polynomial $p(\cdot)$ such that $\mathbb{E}[\mathsf{rew}((\widetilde{P}(x), V^{[x]}))] + \Delta \leq \mathbb{E}[\mathsf{rew}((P(x), V^{[x]}))]$, for $\Delta = 1/p(n)$.

5.2 Our Compiler: AHP to Rational AoK for NP

Consider the indexed relation $\mathcal{R}$ given by triples $(\mathsf{i}, \mathsf{x}, \mathsf{w})$ where the circuit for the relation is given by the index i, x is the instance, and w is the witness. The corresponding *indexed language* $\mathcal{L}(\mathcal{R})$ is the set of pairs (i, x) for which there exists a witness w s.t. $(\mathsf{i}, \mathsf{x}, \mathsf{w}) \in \mathcal{R}$. We construct a compiler that transforms an AHP into a rational argument for NP.

Our compiler is the same as the compiler for P in Section 4. Here, we show that the compiler yields an argument of knowledge for NP, by showing an extractor that satisfies Definition 12. The resulting rational AoK does not perform any cryptographic operations in the indexing step and the prover performs cryptographic operations that grows only with the witness and not the circuit.

Theorem 4. *Let $\mathcal{R}$ be an NP relation. Let* AHP *be an AHP for $\mathcal{R}$ with soundness error ϵ_1, let* PC *be a polynomial commitment scheme with extractability error ϵ_2, and* $\mathsf{rat\text{-}pc}$ *be a rational proof for polynomial evaluation with reward gap Δ. Then, the compiled system is a rational argument of knowledge for $\mathcal{R}$ with reward gap $\Delta(1 - \epsilon_1 - \epsilon_2)$.*

A proof of the above theorem is in Appendix E.3.

[14]Note that this can always be appropriately scaled.

Acknowledgements

The authors thank Mahak Pancholi and Nicola Greco for helpful feedback on early versions of this document. Chaya Ganesh was funded by Protocol Labs through grant 4029342296 (RFP-013). Part of this work was done by Matteo Campanelli and Rosario Gennaro while affiliated with Protocol Labs.

References

[ABC$^+$22] Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi. ECLIPSE: Enhanced compiling method for pedersen-committed zkSNARK engines. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 584–614. Springer, Heidelberg, March 2022. doi:10.1007/978-3-030-97121-2_21.

[AM12] Pablo Daniel Azar and Silvio Micali. Rational proofs. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1017–1028. ACM, 2012.

[AM13] Pablo Daniel Azar and Silvio Micali. Super-efficient rational proofs. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 29–30. ACM, 2013.

[BCG$^+$13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-40084-1_6.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45721-1_24.

[BIS90] David A Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

[CFF$^+$21] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2021. doi:10.1007/978-3-030-92078-4_1.

[CFK22] Matteo Campanelli, Dario Fiore, and Hamidreza Khoshakhlagh. Witness encryption for succinct functional commitments and applications. Cryptology ePrint Archive, Paper 2022/1510, 2022. https://eprint.iacr.org/2022/1510. URL: https://eprint.iacr.org/2022/1510.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019. doi:10.1145/3319535.3339820.

[CG15] Matteo Campanelli and Rosario Gennaro. Sequentially composable rational proofs. In *Decision and Game Theory for Security: 6th International Conference, GameSec 2015, London, UK, November 4-5, 2015, Proceedings 6*, pages 270–288. Springer, 2015.

[CG17] Matteo Campanelli and Rosario Gennaro. Efficient rational proofs for space bounded computations. In Stefan Rass, Bo An, Christopher Kiekintveld, Fei

Fang, and Stefan Schauer, editors, *Decision and Game Theory for Security*, pages 53–73, Cham, 2017. Springer International Publishing.

[CG18] Matteo Campanelli and Rosario Gennaro. Fine-grained secure computation. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 66–97. Springer, Heidelberg, November 2018. doi:10.1007/978-3-030-03810-6_3.

[CGG+23] Matteo Campanelli, Nicolas Gailly, Rosario Gennaro, Philipp Jovanovic, Mara Mihali, and Justin Thaler. Testudo: Linear time prover snarks with constant size proofs and square root size universal setup. In *Progress in Cryptology–LATINCRYPT 2023: 8th International Conference on Cryptology and Information Security in Latin America, LATINCRYPT 2023, Quito, Ecuador, October 3–6, 2023, Proceedings*, pages 331–351. Springer, 2023.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45721-1_26.

[ECK+23] Jens Ernstberger, Stefanos Chaliasos, George Kadianakis, Sebastian Steinhorst, Philipp Jovanovic, Arthur Gervais, Benjamin Livshits, and Michele Orrù. zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks. Cryptology ePrint Archive, Paper 2023/1503, 2023. https://eprint.iacr.org/2023/1503. URL: https://eprint.iacr.org/2023/1503.

[FFG+16] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1304–1316. ACM Press, October 2016. doi:10.1145/2976749.2978368.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, August 2005. doi:10.1007/11535218_10.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96881-0_2.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_37.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013. doi:10.1145/2488608.2488667.

[GHRV14] Siyao Guo, Pavel Hubácek, Alon Rosen, and Margarita Vald. Rational arguments: single round delegation with sublinear verification. In Moni Naor, editor, *ITCS 2014*, pages 523–540. ACM, January 2014. doi:10.1145/2554797.2554845.

[GHRV16] Siyao Guo, Pavel Hubácek, Alon Rosen, and Margarita Vald. Rational sumchecks. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 319–351. Springer, Heidelberg, January 2016. doi:10.1007/978-3-662-49099-0_12.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 113–122. ACM Press, May 2008. doi:10.1145/1374376.1374396.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_11.

[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. https://eprint.iacr.org/2019/953.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992. doi:10.1145/129712.129782.

[KPV22] Assimakis A. Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitments. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1725–1737. ACM Press, November 2022. doi:10.1145/3548606.3560657.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In David B. Shmoys, editor, *46th ACM STOC*, pages 485–494. ACM Press, May / June 2014. doi:10.1145/2591796.2591809.

[KWA$^+$01] Eric Korpela, Dan Werthimer, David Anderson, Jeff Cobb, and Matt Leboisky. Seti@ home-massively distributed computing for seti. *Computing in science & engineering*, 3(1):78–83, 2001.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010. doi:10.1007/978-3-642-17373-8_11.

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 1–34. Springer, Heidelberg, November 2021. doi:10.1007/978-3-030-90453-1_1.

[LSTW21] Jonathan Lee, Srinath Setty, Justin Thaler, and Riad Wahby. Linear-time and post-quantum zero-knowledge SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/030, 2021. https://eprint.iacr.org/2021/030.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019. doi:10.1145/3319535.3339817.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. doi:10.1109/SFCS.1994.365746.

[P$^+$10] Vijay Pande et al. Folding@ home. *Distributed Computing*, 2010.

[RVW13] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 793–802. ACM Press, June 2013. doi:10.1145/2488608.2488709.

[RZ21] Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Heidelberg. doi:10.1007/978-3-030-84242-0_27.

[WTs$^+$18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018. doi:10.1109/SP.2018.00060.

[WYX$^+$21] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 501–518. USENIX Association, August 2021.

Supplementary Material

A Uniform Families of Circuits

Definition 13 ([BIS90]). A circuit family is said to be DLOGTIME-uniform if the connection language

$$\{(1^n, g, h, i) : h \text{ is the } i^{\text{th}} \text{ input gate to } g \text{ in } C_n\} \cup \{(1^n, g, t) : g \text{ is a gate of type } t \text{ in } C_n\}$$

is computable in logarithmic time.

B The Rational Proof from [CG15]

The following protocol works for functions $f : \{0,1\}^n \to \{0,1\}^n$ expressed by an arithmetic circuit $\mathcal{C}$ of size C and depth d and fan-in 2, given as a common input to both Prover and Verifier together with the input x.

Intuitively the idea is for the Prover to provide the Verifier with the output value y and its two "children" y_L, y_R in the gate, i.e. the two input values of the last output gate G. The Verifier checks that $G(y_L, y_R) = y$, and then asks the Prover to verify that y_L or y_R (chosen a random) is correct, by recursing on the above test. The protocol description follows.

1. The Prover evaluates the circuit on x and send the output value y_1 to the Verifier.
2. The Verifier identifies the root gate g_1 and then invokes $\mathit{Round}(1, g_1, y_1)$.

The procedure $\mathit{Round}(i, g_i, y_i)$ is defined for $1 \le i \le d$ as follows:

1. The Prover sends the value of the input wires z_i^0 and z_i^1 of g_i to the Verifier.
2. The Verifiers performs the following
   - Check that y_i is the result of the operation of gate g_i on inputs z_i^0 and z_i^1. If not **STOP** and pay a reward of 0.
   - If $i = d$ (i.e. if the inputs to g_i are input wires), check that the values of z_i^0 and z_i^1 are equal to the corresponding bits of x. Pay reward R to Merlin if this is the case, nothing otherwise.
   - If $i < d$, choose a random bit b, send it to Merlin and invoke $\mathit{Round}(i+1, g_{i+1}^b, z_i^b)$ where g_{i+1}^b is the child gate of g_i whose output is z_i^b.

Efficiency The protocol runs at most in d rounds. In each round, the Prover sends a constant number of bits representing the values of specific input and output wires; The Verifier sends at most one bit per round, the choice of the child gate. Thus the communication complexity is $O(d)$ bits.

The computation of the Verifier in each round is: (i) computing the result of a gate and checking for bit equality; (ii) sampling a child. Gate operations and equality are $O(1)$ per round. We assume our circuits are T-uniform, which allows the Verifier to select the correct gate in time $T(n)$ Thus the Verifier runs in $O(r \cdot T(n))$ with $r = O(\log C)$.

C Discussion on Incentives in Rational Arguments with Extraction

Our notion of rational arguments with extraction is meant to incentivize a party to *know* a witness before engaging in the protocol (for example by searching for it). There are some points of care when applying this notion. For parties to be incentivized to search for a witness, it should not be too costly to do so. Naturally, it should somewhat be feasible for them to search the witness. This is not the case for *all* parties. For parties, for whom it is simply infeasible to search for a witness, the protocol should incentivize them simply not to run the protocol.

In this section we discuss these scenarios more in detail and give an intuition of how they could be dealt with. We will distinguish between two types of parties.

C.1 Parties for whom searching for a witness is feasible

For some of the parties searching for a witness will be feasible, although at some cost. We envision these parties to have significant hardware and bandwidth resources. The witness may be one for a moderately hard NP relation or may require some work that is not strictly computational in flavor, e.g. it may be easy to find once downloaded the whole history of a blockchain.

For this type of parties we need to ensure that incentives are such that it should be more profitable to be honest rather than not:

$$\mathsf{rew}(\text{honest}) - \mathsf{cost}(\text{search}) - \mathsf{cost}(\text{honest\_run}) > \mathsf{rew}(\text{dishonest}) - \mathsf{cost}(\text{dishonest\_run})$$

The honest reward/costs involve:

- $\mathsf{rew}(\text{honest})$: the reward earned by the honest party using a valid witness;
- $\mathsf{cost}(\text{search})$: the cost for a party to search for valid witness;
- $\mathsf{cost}(\text{honest\_run})$: the cost of running the protocol honestly.

The dishonest reward/costs involve:

- $\mathsf{rew}(\text{dishonest})$: the reward earned by a party not using a valid witness;
- $\mathsf{cost}(\text{dishonest\_run})$: the cost of running the protocol not using a valid witness.

In the inequality above, the cost of searching for the witness is application-dependent and in general it cannot be controlled. The rewards (honest and dishonest) can be controlled in several ways. The reward for the honest prover can be increased as needed in the building blocks in the construction in Section 5.2. The reward gap (how much loss the dishonest prover is incurring reward-wise) depends on the underlying building blocks but can be increased for example as described in Appendix D. The remaining costs are those for running the protocol (honestly or dishonestly). The dishonest cost should be high enough so to disincentivize parties to execute the protocol without a witness. See next section for some approaches for increasing the costs of running the protocol.

C.2 Parties for whom searching for a witness is unfeasible

The second set of parties we consider are those for whom it may be too costly to search for the witness. For these type of parties we may want to disincentivize running the protocol altogether. This is incentive is outside the model we formalized in Definition 12. It can, however, be captured by the following inequality:

$$\mathsf{rew}(\text{dishonest}) - \mathsf{cost}(\text{dishonest\_run}) < 0$$

The above can be read as: it should be unprofitable for a party to run the protocol if they do not expect any reward.

In order for the inequality above to true, we may have to increase the cost of running the protocol. We believe that outer applications can handle disincentivizing provers who do not know the witness.

A heuristic approach to this may for example come from a proof-of-work-like approach, for instance from Fischlin's transform [Fis05]. The latter allows to compile public-coin protocols into non-interactive ones. An intuition about how Fischlin's transform works is that it requires the prover to compute a challenge c applying the random oracle on several inputs until a certain number of leading zeros is obtained (this number depends on a chosen hardness parameter). More specifically, if t is the transcript so far and r is a string on which the prover is iterating then a challenge can be obtained by computing $H(t||r)$. Since this requires several attempts, this increases the cost of executing the protocol (for both honest and dishonest parties). This can be a heuristic approach to disincentivizing parties without the witness. We stress that the security of applying Fischlin's transform to produce non-interactive rational arguments is still an open problem.

Other approaches are possible, of course. Emulating proofs of work to increase the work of the prover has been used for example in Starkware[15].

D Reducing the Reward Gap without Losing Sublinearity

It is possible to improve the space gap of a rational proof/argument by parallel composition. Here we discuss how to reduce the reward without losing the verifier's sublinearity. In particular we discuss how to achieve this for reward gaps of the form $\frac{1}{\sqrt[k]{T}}$ where k is a constant and T is roughly the size of the computation. What does this mean concretely? It means that, for example, for a computation with $T \approx 2^{20}$ constraints and for $k = 3$ we can make a cheating prover lose at least $\frac{1}{\sqrt[3]{T}} \approx 5\%$ of the honest reward in expectation. Below we discuss the parameters for which this is possible.

It is possible to show that for the instantiation of our constructions that we consider (which use [CG15]) the reward gap is at least

$$1 - \left(1 - \frac{1}{T}\right)^r$$

where r is the number of repetitions. Now suppose we want to make sure that this reward gap is at least ϵ. We want to find a lower bound for r for which this is possible. At the same time, to preserve sublinearity, we will make sure that $r = o(n)$ where n is the size of the public input. We set the inequality

$$1 - \left(1 - \frac{1}{T}\right)^r \geq \epsilon$$

and by simple algebraic manipulations we obtain that this is true for

$$r \geq \frac{\log\left(1-\epsilon\right)}{\log\left(1 - \frac{1}{T}\right)}$$

.

By using Maclaurin expansion we can apply the approximation $\log(1-x) \approx x$ and obtain $r \approx \epsilon \cdot T$. In order to obtain a sublinear r we need to consider non-constant ϵ-s. Thus, let $k = O(1)$ such that $\epsilon = \frac{1}{\sqrt[k]{T}}$. Let us assume that our (polynomial-time) computation is of size n^c for a constant c. Then, fixed such a computation, we can obtain $r = o(n)$ if we choose k such that $c < \frac{k}{k-1}$.

What this means is that, for example, we can obtain a sublinear number of repetitions for a reward gap $\frac{1}{\sqrt[3]{T}}$ for any computation of size at most $n^{1.5}$.

[15]See https://a16zcrypto.com/posts/article/snark-security-and-performance/.

E Additional Proofs

E.1 Proof of Lemma 1

Proof. To see why the first property holds, consider z the output of the circuit. We can observe that:

$$\begin{aligned} z =& a_0 + \sum_{i \in [2^\ell]} s_i \\ =& a_0 + \sum_{i \in [2^\ell]} a_i \cdot y_i \\ =& a_0 + \sum_{i \in [2^\ell]} a_i \cdot \prod_{j \in [\ell] : b_j(i) = 1} t^{2^j} \\ =& a_0 + \sum_{i \in [2^\ell]} a_i \cdot t^i \\ =& f(t) \end{aligned}$$

We provide an intuition of why the second property holds. What we need to show is that the wiring of the circuit can be decided in logarithmic time (or, in time linear in the size of the wire indices). Let us observe the structure of the circuit. It is composed of *trees* of addition gates (computation of z) or multiplication gates (computation of y_i-s and computation of s_i-s, which can be seen as a multiplication tree of size 1). We observe that the wiring structure of trees is very easy to decide (we give more details below) and that these trees are arranged homogeneously around the circuit and thus there is little overhead besides deciding their internal structure. To see why they are distributed "homogeneously", notice that we have: 2^ℓ parallel multiplication trees, each with a multiplication gate on top; the results of these are then aggregated through a final addition tree on top.

To see why the internal structure of trees can be decided in logarithmic time, given (a, b, c):

- wire a is the left sibling of wire b iff their bit representation is such that $a = X0$, $b = X1$ where X is a bitstring prefix.
- wire c is the child of a and b iff the bit representation of c is X (for X defined as in the previous item, i.e. the right shift by one of the bit representation of a and b).

The only additional point for deciding wiring is for the product tree when computing y_i, in particular the condition $b_j(i) = 1$ in the product. This is required only for the leaves of said tree can easily be decided in logarithmic time given the definition of the predicate b_j (if such predicate holds the leaf should correspond to the respective input wire, otherwise it should correspond to an input wire for the constant 1). $\square$

E.2 Proof of Theorem 3

Proof. Since the reward gap is noticeable, we have that $\Delta(n) > 1/p(n)$ for some polynomial $p(\cdot)$ for large enough n. We show how to transform (P, V) for L with noticeable $\Delta(n)$ into an interactive proof for L with noticeable completeness-soundness gap. Let x be an instance to be decided. The following procedure M decides x: Evaluate $\mathsf{rew}(x, \pi)$ where π is the transcript for all possible $2^{\log n}$ values of π. Repeat this k times to estimate expected values. Let $\pi_{\mathsf{acc}} \in \{0,1\}^{\log n}$ be the set of transcripts π where the prover sends 1 that is, such that $\mathsf{out}((P, V)(x) = 1$ and π_{rej} be the set of transcripts such that $\mathsf{out}((P, V)(x) = 0$. If $\max_{\pi \in \pi_{\mathsf{acc}}} \mathbb{E}[\mathsf{rew}(x, \pi)] > \max_{\pi \in \pi_{\mathsf{rej}}} \mathbb{E}[\mathsf{rew}(x, \pi)]$, then output 1, otherwise output 0. If we set $k = \mathsf{poly}(p)$, by Hoeffding's inequality, we can show that $\mathbb{E}[\mathsf{rew}(x, \pi)]$ for each π is estimated within $\varepsilon = \Delta/3$ with high probability, so that M decides x correctly with probability $> 2/3$. Finally, note that M is randomized and runs in polynomial time. Thus, L is decided by a PPT machine. $\square$

E.3 Proof of Theorem 4

Proof. Let $\mathsf{AHP}^{\{p_{i,j}\}}$ be the starting AHP with polynomial oracles. Consider a hybrid intermediate oracle argument system $\mathsf{AHP}^{\{p_{0,j}\}}$ where the polynomial oracles of $\{p_{i,j}\}$, for $i \neq 0$ are realized via a cyptographic PCS PC, and the polynomials $\{p_{0,j}\}$ are still available as oracles. Suppose prover $\tilde{\mathsf{P}}$ of $\mathsf{AHP}^{\{p_{0,j}\}}$ convinces V of instance (i, x) in $L_{\mathcal{R}}$. We construct an adversary $\mathcal{A}_{\mathsf{pc}}$ against the extractability game of PC. $\mathcal{A}_{\mathsf{PC}}$ receives the commitment key ck, and random coins and internally invokes $\tilde{\mathsf{P}}$ to obtain a set of commitments $\{C_{i,j}\}_{i\in[k],j\in[s(i)]}$. We now invoke the extractor $\mathcal{E}_{\mathsf{PC}}$, which given the same input as $\mathcal{A}_{\mathsf{PC}}$ outputs a set of polynomials $\tilde{p} = \{p_{i,j}\}_{i\in[k],j\in[s(i)]}$. Decode the witness $\hat{\mathsf{w}}$ from the witness-carrying polynomial and output $\hat{\mathsf{w}}$.

We now argue that $(\mathsf{i}, \mathsf{x}, \hat{\mathsf{w}}) \in \mathcal{R}$. If $\mathcal{E}_{\mathsf{pc}}$ fails (in the hybrid argument system $\mathsf{AHP}^{\{p_{0,j}\}}$), then the extracted polynomials are inconsistent with the alleged evaluations ($\tilde{p}(\mathbf{z}) \neq \mathbf{v}$), and $\mathcal{A}_{\mathsf{PC}}$ wins the extractability game, which happens with probability ϵ_2. Hence, except with probability ϵ_2, $\tilde{p}(\mathbf{z}) = \mathbf{v}$. Given that the evaluations are valid w.r.t the set of polynomials $\tilde{p}$, V accepts when AHP verifier accepts. Since the soundness error of the AHP is ϵ_1, the probability that V accepts $(f, x, \tilde{y}) \notin L_{\mathcal{R}}$ is $\epsilon_1 + \epsilon_2$, which is negligible.

Now consider the argument system where $\mathsf{rat\text{-}pc}$ is used to instantiate the oracles $\{p_{0,j}\}$. The output of $\mathcal{E}_{\mathsf{pc}}$ is not a valid witness if $\tilde{\mathsf{P}}$ lies in $\mathsf{rat\text{-}pc}$ and therefore the extracted witness from $\mathcal{E}_{\mathsf{pc}}$ is not a witness consistent with the input/index polynomial in $\{p_{0,j}\}$. Let $\epsilon_{\tilde{\mathsf{P}}}$ be the probability with which $\tilde{\mathsf{P}}$ cheats in $\mathsf{rat\text{-}pc}$. The expected reward gap of $\tilde{\mathsf{P}}$ is $\geq \epsilon_{\tilde{\mathsf{P}}} \cdot \Delta$. Therefore, if the extractor succeeds with probablity at most $1 - (\epsilon_{\tilde{\mathsf{P}}} + \mathsf{negl})$, the reward gap of the prover $\tilde{\mathsf{P}}$ is $\geq \epsilon_{\tilde{\mathsf{P}}} \cdot \Delta$. Since Δ is noticeable, when extraction fails with probability $\geq 1/\mathsf{poly}$, that is, $\epsilon_{\tilde{\mathsf{P}}} \geq 1/\mathsf{poly}$, then the reward gap of the prover is $\geq 1/\mathsf{poly}$. $\square$

F Candidate AHPs

We state as corollaries concrete instantiataions obtained by applying our compiler from Figure 3.

Corollary 1. *For an* NP *relation with arithmetic complexity n, public input size $|x|$, Figure 3 instantiated with Dory PCS and PLONK's AHP yields a rational argument of knowledge with full indexing as per Definition 12 and satisfies the following properties:*

- *Preprocessing complexity: $\tilde{O}(n)$ non-cryptographic operations*
- *Communication complexity: $O(\log n)$*
- *Verification complexity: $O(\log n)$ cryptographic + $O(|x|)$ non-cryptographic*
- *Prover complexity: $O(n \log n)$ cryptographic + $O(|x|)$ non-cryptographic*

Corollary 2. *For an* NP *relation with R1CS complexity n, public input size $|x|$, Figure 3 instantiated with Dory PCS and Marlin's AHP yields a rational argument of knowledge with full indexing as per Definition 12 and satisfies the following properties:*

- *Preprocessing complexity: $\tilde{O}(n)$ non-cryptographic operations*
- *Communication complexity: $O(\log n)$*
- *Verification complexity: $O(\log n)$ cryptographic + $O(|x|)$ non-cryptographic*
- *Prover complexity: $O(n \log n)$ cryptographic + $O(|x|)$ non-cryptographic*

```
$\mathsf{PolyEvalPreproc}_{\mathsf{Lag}}(f', \mathbf{u})$

  Parse $\mathbf{u}$ as $(u_1, \ldots, u_\ell)$
  Parse $f'$ as a tuple of $(x, v)$ pairs such that $x = x_1 || \cdots || x_d$ for $x_i \in \mathbb{H}$
  Compute $f(X) = \sum v_i L_{x_i}(X) = \sum_{i=0}^{d} f_i X^i$, where $d = 2^\ell$
  for $i = 1, \ldots, 2^\ell$ (in parallel)
    $y_i \leftarrow \prod_{j \in [\ell] : b_j(i) = 1} u_j$
      where $b_j(i)$ is the $j$-th bit of $i$ for each $j$
    $s_i \leftarrow f_i \cdot y_i$
  endfor
  $z \leftarrow a_0 + \sum_{i \in [2^\ell]} s_i$
  return $z$
```

Figure 4: An arithmetic circuit with logarithmic depth for polynomial evaluation given input in Lagrange basis

We now give a rational proof for polynomial evaluation where the polynomial is given in Lagrange basis instead of monomial basis (Section F.1). This is a straightforward adaptation of the protocol in Section 3.2. Then, in subsequent sections, we show how to apply our compiler to Marlin, PLONK and Sonic AHPs. For the sake of completeness, we present the AHPs (the underlying AHPs of Marlin, PLONK and Sonic are abstracted out in [ABC+22], which we reproduce). We then observe that it satisfies the properties required by our compiler – the decision algorithm of the AHP verifier uses the public input only by accessing an encoding of it. This encoding is compatible with the oracle input of the verifier in our rational proof for polynomial evaluation. That is, the public input is encoded either as a coefficient vector (Sonic) of a polynomial or as point-value pairs that intrepolates a polynomial (Marlin and PLONK).

F.1 Rational proof for polynomial evaluation in Lagrange basis

Let $\mathbb{H}$ be some domain, $\{L_a(X)\}_{a\in\mathbb{H}}$ be the *Lagrange basis polynomials* of degree less than d such that $L_a(a) = 1$ and $L_a(a') = 1$ for $a' \in \mathbb{H} \setminus \{a\}$.

The circuit $\mathsf{PolyEvalPreproc}_{\mathsf{Lag}}$ (Figure 4) first interpolates a polynomial given its evaluation at $d-1$ points in the domain, and then performs evaluation as in $\mathsf{PolyEvalPreproc}$. We now note that this is a highly parallel arithmetic circuit since interpolation via the butterfly network is parallel. The sub-circuit that converts point-value representation to coefficient representation has depth $\log d$ and size $d \log d$, and the sub-circuit for evaluation given coefficient representation has depth $O(\log d)$ as in Section 3.2. We can apply the rational proof for parallel circuits from [CG15]. Therefore, Theorem 1 (there exists a rational proof for polynomial evaluation where the verifier runs in time $O(\log(d))$, and the reward gap is noticeable) holds when the polynomial is given in Lagrange basis. The rational proof for polynomial evaluation given a polynomial in point-value representation is in Figure 5.

Rational proof for $F_{\text{poly-Lag},f}(t)$:

- Setup:
  - Do nothing
- $\mathsf{idx}(f)$:
  - return a description of f as a tuple of point-value pairs $\{(y_i, v_i)\}_{i\in[d]}$ where each y_i is in some known domain $\mathbb{H}$, and $v_i = f(y_i)$. $f := (\mathbf{y}, \mathbf{v})$.
- Protocol $P(f,t) \leftrightarrow V^{[f]}(t)$:
  - Both parties compute $\mathbf{u} = (t, t^2, t^4, \ldots, t^{2^{\ell-1}}, t^{2^\ell})$ by repeated squaring where $\mathsf{deg}(f) = 2^\ell$
  - The two parties then proceed by running $\mathsf{RPPar}.P_F \leftrightarrow \mathsf{RPPar}.V_F^{[x]}$ where the circuit $F := \mathsf{PolyEvalPreproc}_{\mathsf{Lag}}$ and input $x := (f, \mathbf{u})$.
- Reward:
  - Return reward from $\mathsf{RPPar.rew}^{[x]}$

Figure 5: Rational proof for polynomial evaluation given polynomial in Lagrange basis. The protocol RPPar is from the statement of Lemma 2.

F.2 Marlin AHP

In this section we show how to apply our compiler to Marlin. We show how to achieve a sublinear decision algorithm for the verifier, identify the WCPs and show how it encodes the witness vector in the AHP.

Notation. For a finite field $\mathbb{F}$ and a subset $\mathbb{S} \subseteq \mathbb{F}$, $v_{\mathbb{S}}(X)$ denotes the vanishing polynomial of $\mathbb{S}$ that is the unique non-zero monic polynomial of degree at most $|\mathbb{S}|$ that is zero everywhere on $\mathbb{S}$. $\mathbb{F}^{\mathbb{S}}$ denotes the set of vectors indexed by elements in a finite set $\mathbb{S}$. For a function $f : \mathbb{S} \to \mathbb{F}$, we denote by $\hat{f}$, the univariate polynomial over $\mathbb{F}$ with degree less than $|\mathbb{S}|$ that agrees with f, that is, $\hat{f}(a) = f(a)$ for all $a \in \mathbb{S}$. The polynomial $\hat{f}$ can be expressed as

$$\hat{f}(X) = \sum_{a\in\mathbb{S}} f(a) \cdot L_{a,\mathbb{S}}(X)$$

where $\{L_{a,\mathbb{S}}(X)\}_{a\in\mathbb{S}}$ are the *Lagrange basis polynomials* of degree less than $|\mathbb{S}|$ such that $L_{a,\mathbb{S}}(a) = 1$ and $L_{a,\mathbb{S}}(a') = 1$ for $a' \in \mathbb{S} \setminus \{a\}$. For an $n \times n$ matrix M with rows/columns indexed by elements of $\mathbb{S}$, $\hat{M}(X, Y)$ denotes the polynomial of individual degree less than n, where $\hat{M}(s,t)$ is the (s,t)th entry of M for all $s, t \in \mathbb{S}$.

Consider the bivariate polynomial $u_{\mathbb{S}}(X, Y)$

$$u_{\mathbb{S}}(X, Y) := \frac{v_{\mathbb{S}}(X) - v_{\mathbb{S}}(Y)}{X - Y}$$

such that $u_{\mathbb{S}}(X, X) = |\mathbb{S}|X^{|\mathbb{S}|-1}$ is the formal derivative of the vanishing polynomial $v_{\mathbb{S}}(X)$. $u_{\mathbb{S}}(X, Y)$ vanishes on the square $\mathbb{S} \times \mathbb{S}$, except on the diagonal, where it takes $u_{\mathbb{S}}(a, a)_{a\in\mathbb{S}}$.

R1CS Constraint system. R1CS (Rank-1 constraint satisfiability) indexed relation defined by the set of tuples $(\mathsf{i}, \mathsf{x}, \mathsf{w}) = \big((\mathbb{F}, \mathbb{H}, \mathbb{K}, A, B, C), x, w\big)$, where $\mathbb{F}$ is a finite field, $\mathbb{H}$ and $\mathbb{K}$ are subsets of $\mathbb{F}$, such that $n = |\mathbb{H}|$ and $m = |\mathbb{K}|$, A, B, C are $\mathbb{H} \times \mathbb{H}$ matrices over $\mathbb{F}$ with $|\mathbb{K}| \geq \max\{\|A\|, \|B\|, \|C\|\}$, and $z := (x, w)$ is a vector in $\mathbb{F}^{\mathbb{H}}$ such that $Az \circ Bz = Cz$.

As in Marlin [CHM+20], let $\mathbb{H}$ and $\mathbb{K}$ be multiplicative subgroups of $\mathbb{F}$. We assume efficiently computable bijections $\phi_{\mathbb{H}} : \mathbb{H} \to [n]$ and $\phi_{\mathbb{K}} : \mathbb{K} \to [m]$, and denote the first l

elements in $\mathbb{H}$ and the remaining elements, via sets $\mathbb{H}[\leq l] \coloneqq \{a \in \mathbb{H} : 1 \leq \phi_{\mathbb{H}}(a) \leq l\}$ and $\mathbb{H}[> l] \coloneqq \{a \in \mathbb{H} : l < \phi_{\mathbb{H}}(a) \leq n\}$ respectively. We then denote the first part of the vector z as the public component $x \in \mathbb{F}^{\mathbb{H}[\leq l]}$ and the second part as witness component $w \in \mathbb{F}^{\mathbb{H}[>l]}$.

AHP Marlin's AHP is formally described in Fig. 6. In the preprocessing phase, the indexer $\mathcal{I}$ receives as input a field $\mathbb{F}$, subsets $\mathbb{H}, \mathbb{K}$ of $\mathbb{F}$, and matrices $A, B, C \in \mathbb{F}^{\mathbb{H}\times\mathbb{H}}$ representing the R1CS instance. The output of the preprocessing phase is three univariate polynomials $\{\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M\}$ of degree less than $|\mathbb{K}|$ for each matrix $M \in \{A, B, C\}$, such that the following polynomial is a low-degree extension of M.

$$\hat{M}(X, Y) \coloneqq \sum_{k\in\mathbb{K}} u_{\mathbb{H}}(X, \hat{\mathsf{row}}_M(k)) u_{\mathbb{H}}(Y, \hat{\mathsf{col}}_M(k)) \hat{\mathsf{val}}_M(k)$$

The polynomials $\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M$ are the unique low-degree extensions of the functions $\mathsf{row}_M, \mathsf{col}_M, \mathsf{val}_M : K \to \mathbb{F}$ that denote the row index, column index and value of the non-zero entries of the matrix M respectively. Let $\hat{M}(X, Y)$ be the unique low-degree extension of M that agrees with the matrix M everywhere on the domain $\mathbb{H} \times \mathbb{H}$. The prover $\mathcal{P}$ receives as input the instance $x \in \mathbb{F}^{\mathbb{H}[\leq l]}$, a witness $w \in \mathbb{F}^{\mathbb{H}[>l]}$. The verifier $\mathcal{V}$ receives as input x, and obtains oracle access to the nine polynomials output at the end of the preprocessing phase.

Let $\hat{x}(X) \in \mathbb{F}_{<l}[X]$ and $\hat{w}(X) \in \mathbb{F}_{\leq n-l}[X]$ be polynomials that agree with the statement x on $\mathbb{H}[\leq l]$, and with the shifted witness on $\mathbb{H}[> l]$ respectively. These polynomials are defined as follows:

$$\hat{x}(X) \coloneqq \sum_{a\in\mathbb{H}[\leq l]} x(a) \cdot L_{a,\mathbb{H}[\leq l]}(X)$$

$$\hat{w}(X) \coloneqq \sum_{a\in\mathbb{H}[>l]} \left(\frac{w(a) - \hat{x}(a)}{v_{\mathbb{H}[\leq l]}(a)}\right) \cdot L_{a,\mathbb{H}[>l]}(X)$$

Let $z \coloneqq (x, w)$ denote the full assignment. Then the polynomial

$$\hat{z}(X) \coloneqq \hat{w}(X) \cdot v_{\mathbb{H}[\leq l]}(X) + \hat{x}(X)$$

agrees with z on $\mathbb{H}$. The prover computes the linear combinations $z_A \coloneqq Az, z_B \coloneqq Bz$, $z_C \coloneqq Cz$, and sets polynomials $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X) \in \mathbb{F}_{\leq n}[X]$.

Sublinear Decision algorithm and WCP. In $\mathsf{AHP}_{\mathsf{Marlin}}$(Fig. 6), the prover $\mathcal{P}$ receives as input the instance $x \in \mathbb{F}^{\mathbb{H}[\leq l]}$, a witness $w \in \mathbb{F}^{\mathbb{H}[>l]}$. Note that the verifier $\mathcal{V}$ receives as input x, and obtains oracle access to the nine polynomials output at the end of the preprocessing phase.

$\mathcal{V}$ evaluates $\hat{z}$ at point γ using $O(|x|)$ operations by querying $\hat{w}$ at γ and computing the expression $\hat{w}(\gamma) \cdot v_{\mathbb{H}[\leq l]}(\gamma) + \hat{x}(\gamma)$. In our compiled argument, this $O(|x|)$ computation is done by the prover, and the verifier only checks a rational proof of this computation. Note that in the Marlin AHP decision algorithm, the only step that needs $O(|x|)$ computation is step 3 reproduced below:

$$s(\beta_1) + u_{\mathbb{H}}(\alpha, \beta_1)(\sum_M \eta_M \hat{z}_M(\beta_1)) - \sigma_2 \hat{z}(\beta_1) = h_1(\beta_1) v_{\mathbb{H}}(\beta_1) + \beta_1 g_1(\beta_1) + \sigma_1/|\mathbb{H}|$$

Here, $\mathsf{D}_{\mathcal{V}}$ needs to evaluate $u_{\mathbb{H}}$ at (α, β_1), and $v_{\mathbb{H}}$ and $\hat{z}$ at β_1. $u_{\mathbb{H}}$ and $v_{\mathbb{H}}$ can be evaluated in $O(\log|\mathbb{H}|)$ operations. $\hat{z}$ which is the only online linear operation can be evaluated given oracle access to $\hat{x}$ (as point-value pairs $(a, x(a)_{a\in\mathbb{H}})$) by querying $\hat{w}$, $\hat{x}$ and $v_{\mathbb{H}}$ at γ and computing $\hat{z}(\gamma) = \hat{w}(\gamma) \cdot v_{\mathbb{H}[\leq l]}(\gamma) + \hat{x}(\gamma)$.

Protocol 1: Protocol $\mathsf{AHP}_{\mathsf{Marlin}}$

Offline phase. The indexer $\mathcal{I}$ is given as input a field $\mathbb{F} \in \mathcal{F}$, subsets $\mathbb{H}, \mathbb{K}$ of $\mathbb{F}$, and matrices $A, B, C \in \mathbb{F}^{n\times n}$ representing the R1CS instance, and outputs three univariate polynomial oracles $\{\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M\}$ of degree less than $|\mathbb{K}|$ for each matrix $M \in A, B, C$, such that the following polynomial is a low-degree extension of M.

$$\hat{M}(X,Y) := \sum_{k\in\mathbb{K}} u_{\mathbb{H}}(X, \hat{\mathsf{row}}_M(k))u_{\mathbb{H}}(Y, \hat{\mathsf{col}}_M(k))\hat{\mathsf{val}}_M(k)$$

Input. $\mathcal{P}$ receives $(\mathbb{F}, \mathbb{H}, \mathbb{K}, A, B, C, \mathsf{i}, x, w)$, and $\mathcal{V}$ receives $(\mathbb{F}, \mathbb{H}, \mathbb{K}, x)$ and oracle access to the nine polynomials output by $\mathcal{I}(\mathbb{F}, \mathsf{i})$.

Online phase: round 1. $\mathcal{P}$ sends the oracle polynomials $\hat{w}(X) \in \mathbb{F}_{\leq n-l}[X], h_0(X), \hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X) \in \mathbb{F}_{\leq n}[X]$. It samples a random $s(X) \in \mathbb{F}_{<2n}[X]$ and sends polynomial oracle $s(X)$ together with $\sigma_1 \in \mathbb{F}$ where $\sigma_1 := \sum_{a\in\mathbb{H}} s(a)$, and $\hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = h_0(X)v_{\mathbb{H}}(X)$.

Online phase: round 2. On receiving challenges $\alpha, \eta_A, \eta_B, \eta_C \in \mathbb{F}$ from $\mathcal{V}$, $\mathcal{P}$ sends oracle polynomials $g_1(X) \in \mathbb{F}_{<n-1}[X]$, $h_1(X) \in \mathbb{F}_{<n}[X]$ to $\mathcal{V}$, where

$$s(X) + u_{\mathbb{H}}(\alpha, X)\Bigg(\sum_{M\in\{A,B,C\}} \eta_M \hat{z}_M(X)\Bigg) - \Bigg(\sum_{M\in\{A,B,C\}} \eta_M r_M(\alpha, X)\Bigg)\hat{z}(X)$$
$$= h_1(X)v_{\mathbb{H}}(X) + Xg_1(X) + \sigma_1/|\mathbb{H}|$$

Online phase: round 3. On receiving challenge $\beta_1 \in \mathbb{F}$ from the $\mathcal{V}$, $\mathcal{P}$ sends oracle polynomials $g_2(X), h_2(X) \in \mathbb{F}_{<n-1}[X]$ and $\sigma_2 \in \mathbb{F}$ to $\mathcal{V}$, where

$$\sigma_2 := \sum_{k\in\mathbb{H}} u_{\mathbb{H}}(\alpha, k) \sum_{M\in\{A,B,C\}} \eta_M \hat{M}(k, \beta_1)$$

$$u_{\mathbb{H}}(\alpha, X) \sum_{M\in\{A,B,C\}} \eta_M \hat{M}(X, \beta_1) = h_2(X)v_{\mathbb{H}}(X) + Xg_2(X) + \sigma_2/|\mathbb{H}|$$

Online phase: round 4. On receiving challenge $\beta_2 \in \mathbb{F}$ from $\mathcal{V}$, $\mathcal{P}$ sends oracle polynomials $g_3(X) \in \mathbb{F}_{<m-1}[X]$ $h_3(X) \in \mathbb{F}_{<6m-6}[X]$ and $\sigma_3 \in \mathbb{F}$ to $\mathcal{V}$, where where

$$\sigma_3 := \sum_{k\in\mathbb{K}} \sum_{M\in\{A,B,C\}} \eta_M \frac{v_{\mathbb{H}}(\beta_2)v_{\mathbb{H}}(\beta_1)\hat{\mathsf{val}}_M(k)}{(\beta_2 - \hat{\mathsf{row}}_M(k))(\beta_1 - \hat{\mathsf{col}}_M(k))}$$

$$h_3(X)v_{\mathbb{K}}(X) = a(X) - b(X)(Xg_3(X) + \sigma_3/|\mathbb{K}|)$$

$$a(X) = \sum_{M\in\{A,B,C\}} \eta_M v_{\mathbb{H}}(\beta_2)v_{\mathbb{H}}(\beta_1)\hat{\mathsf{val}}_M(X) \prod_{L\in\{A,B,C\}\setminus\{M\}} (\beta_2 - \hat{\mathsf{row}}_L(X))(\beta_1 - \hat{\mathsf{col}}_L(X))$$

$$b(X) = \prod_{M\in\{A,B,C\}} (\beta_2 - \hat{\mathsf{row}}_M(X))(\beta_1 - \hat{\mathsf{col}}_M(X))$$

Query phase. $\mathcal{V}$ queries the oracles $\hat{w}(X), \hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X), h_0(X), s(X), h_1(X), g_1(X)$ at β_1; $h_2(X), g_2(X)$ at β_2; $h_3(X), g_3(X)$ and all offline oracles $\{\hat{\mathsf{row}}_M, \hat{\mathsf{col}}_M, \hat{\mathsf{val}}_M\}$ for each $M \in A, B, C$ at a random query point $\beta_3 \in \mathbb{F}$.

Decision phase $\mathsf{D}_{\mathcal{V}}$. $\mathcal{V}$ accepts if the following tests pass:

1. $h_3(\beta_3)v_{\mathbb{K}}(\beta_3) = a(\beta_3) - b(\beta_3)(\beta_3 g_3(\beta_3) + \sigma_3/|\mathbb{K}|)$
2. $h_2(\beta_2)v_{\mathbb{H}}(\beta_2) + \beta_2 g_2(\beta_2) + \sigma_2/|\mathbb{H}| = u_{\mathbb{H}}(\alpha, \beta_2)\sigma_3$
3. $s(\beta_1) + u_{\mathbb{H}}(\alpha, \beta_1)(\sum_M \eta_M \hat{z}_M(\beta_1)) - \sigma_2\hat{z}(\beta_1) = h_1(\beta_1)v_{\mathbb{H}}(\beta_1) + \beta_1 g_1(\beta_1) + \sigma_1/|\mathbb{H}|$
4. $\hat{z}_A(\beta_1)\hat{z}_B(\beta_1) - \hat{z}_C(\beta_1) = h_0(\beta_1)v_{\mathbb{H}}(\beta_1)$

Figure 6: AHP for $\mathcal{R}_{R1CS}$

F.3 PLONK AHP

Constraint System and AHP. Consider an arithmetic circuit with fan-in two over $\mathbb{F}$, consisting of n gates. The PLONK AHP proves knowledge of left, right and output wire values for every gate $i \in [n]$ in the circuit, such that they are consistent with the constraints determined by the circuit topology. The per-gate constraints are specified by *selector vectors* $\mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_O, \mathbf{q}_M, \mathbf{q}_C \in \mathbb{F}^n$. We call $\mathcal{C} = (n, m, \mathbf{L}, \mathbf{R}, \mathbf{O}, \mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_O, \mathbf{q}_M, \mathbf{q}_C)$ *constraint systems.* $\mathbf{w} \in \mathbb{F}^m$ is said to satisfy the constraint systems $\mathcal{C}$ if for each gate $i \in [n]$

$$(\mathbf{q}_L)_i \cdot w_{\mathbf{L}_i} + (\mathbf{q}_R)_i \cdot w_{\mathbf{R}_i} + (\mathbf{q}_O)_i \cdot w_{\mathbf{O}_i} + (\mathbf{q}_M)_i \cdot w_{\mathbf{L}_i} w_{\mathbf{R}_i} + (\mathbf{q}_C)_i = 0. \tag{3}$$

For the wire values $\mathbf{w} \in \mathbb{F}^m$, $(w_j)_{j \in [l]}$ denote the *public input* and $(w_j)_{j \in [l+1,m]}$ the *private input* respectively. We say $\mathcal{C}$ *is prepared for* l *public inputs* if for each $i \in [l]$, $\mathbf{L}_i = i$, $(\mathbf{q}_L)_i = 1, (\mathbf{q}_R)_i = (\mathbf{q}_M)_i = (\mathbf{q}_R)_i = (\mathbf{q}_C)_i = 0$, i.e., each gate $i \in [l]$ is dedicated for the input wire $j = i \in [l]$ of $\mathbf{w}$. The constraint for an input gate $i \in [l]$ is satisfied by subtracting w_j from the above equation. The relation wrt $\mathcal{C}$ is defined below.

Definition 14 (PLONK indexed relation)**.** The indexed relation $\mathcal{R}_{\mathsf{PLONK}}$ is the set of all triples

$$((\mathbb{F}, n, m, l, \mathbf{L}, \mathbf{R}, \mathbf{O}, \mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_O, \mathbf{q}_M, \mathbf{q}_C), (w_j)_{j \in [l]}, (w_j)_{j \in [l+1,m]})$$

such that

$$\forall i \in [l], (\mathbf{q}_L)_i \cdot w_{\mathbf{L}_i} + (\mathbf{q}_R)_i \cdot w_{\mathbf{R}_i} + (\mathbf{q}_O)_i \cdot w_{\mathbf{O}_i} + (\mathbf{q}_M)_i \cdot w_{\mathbf{L}_i} w_{\mathbf{R}_i} + (\mathbf{q}_C)_i - w_i = 0$$
$$\forall i \in [l+1, n], (\mathbf{q}_L)_i \cdot w_{\mathbf{L}_i} + (\mathbf{q}_R)_i \cdot w_{\mathbf{R}_i} + (\mathbf{q}_O)_i \cdot w_{\mathbf{O}_i} + (\mathbf{q}_M)_i \cdot w_{\mathbf{L}_i} w_{\mathbf{R}_i} + (\mathbf{q}_C)_i = 0$$

PLONK AHP relies on a multiplicative subgroup $\mathbb{H} = \{\zeta, \zeta^2, \ldots, \zeta^n\} \subset \mathbb{F}^*$ generated by the nth primitive root of unity $\zeta \in \mathbb{F}^*$. $v_{\mathbb{H}}(X) = X^n - 1$, an associated vanishing polynomial, splits completely in $\mathbb{F}[X]$, i.e., $X^n - 1 = \prod_{i=1}^{n}(X - \zeta^i)$. We have the corresponding Lagrange basis $L_i(X) \in \mathbb{F}_{<n}[X]$ for $i \in [n]$ such that $L_i(\zeta^i) = 1$ and $L_i(\zeta^j) = 0$ for $j \neq i$.

The selector vectors define polynomials in $\mathbb{F}_{<n}[X]$:

$$q_L(X) = \sum_{i \in [n]} (\mathbf{q}_L)_i \cdot L_i(X) \quad q_R(X) = \sum_{i \in [n]} (\mathbf{q}_R)_i \cdot L_i(X) \quad q_O(X) = \sum_{i \in [n]} (\mathbf{q}_O)_i \cdot L_i(X) \tag{4}$$

$$q_M(X) = \sum_{i \in [n]} (\mathbf{q}_M)_i \cdot L_i(X) \quad q_C(X) = \sum_{i \in [n]} (\mathbf{q}_C)_i \cdot L_i(X) \tag{5}$$

So $q_L(\zeta^i) = (\mathbf{q}_L)_i, q_R(\zeta^i) = (\mathbf{q}_R)_i$ and so on. Now we can define the following polynomials.

$$f_{\mathsf{pub}}(X) = \sum_{i \in [l]} -w_i L_i(X) \quad f_L(X) = \sum_{i \in [n]} w_{\mathbf{L}_i} L_i(X) \quad f_R(X) = \sum_{i \in [n]} w_{\mathbf{R}_i} L_i(X) \quad f_O(X) = \sum_{i \in [n]} w_{\mathbf{O}_i} L_i(X) \tag{6}$$

The gate-by-gate constraint of Eq. (3) can be checked as follows:

$$F_{\mathcal{C}}(X) \coloneqq q_L(X) f_L(X) + q_R f_R(X) + q_O(X) f_O(X) + q_M(X) f_L(X) f_R(X) + q_C(X) + f_{\mathsf{pub}}(X) \tag{7}$$

vanishes at ζ^i for all $i \in [n]$.

The PLONK AHP is shown in Fig. 7 [ABC$^+$22].

Sublinear Decision algorithm and WCP. During the first round of $\mathsf{AHP}_{\mathsf{PLONK}}$, the prover sends the following WCPs encoding both statement and witness $((\mathsf{w}_i)_{i \in [l]}$, $(\mathsf{w}_i)_{i \in [l+1,3n]})$:

$$f_L(X) = \sum_{i \in [n]} \mathsf{w}_i L_i(X) \quad f_R(X) = \sum_{i \in [n]} \mathsf{w}_{n+i} L_i(X) \quad f_O(X) = \sum_{i \in [n]} \mathsf{w}_{2n+i} L_i(X) \tag{14}$$

Note that in the PLONK AHP decision algorithm, the only step that needs $O(|x|)$ computation is in computing $f_{\mathsf{pub}}(X)$ at z: $f_{\mathsf{pub}}(X) = \sum_{i \in [l]} -\mathsf{w}_i L_i(X)$ In our compiled argument, this is done by the prover and the verifier only checks a rational proof of evaluation given oracle access to (w_i, x_i) pairs for $x_i \in \mathbb{H}$.

F.4 Sonic AHP

Vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$ each of length n represent left inputs, right inputs and outputs respectively of the multiplication gates.

$$\mathbf{a} \odot \mathbf{b} = \mathbf{c}$$

Let $\mathbf{u}_q, \mathbf{v}_q, \mathbf{w}_q \in \mathbb{F}^n$ be fixed vectors for the qth linear constraint with instance values $k_q \in \mathbb{F}$. There are Q linear constraints of the form,

$$\mathbf{a} \cdot \mathbf{u}_q + \mathbf{b} \cdot \mathbf{v}_q + \mathbf{c} \cdot \mathbf{w}_q = k_q$$

The n multiplication constraints are compressed into one equation by introducing a formal indeterminate Y.

$$\sum_{i=1}^{n} (a_i b_i - c_i) Y^i = 0 \qquad \sum_{i=1}^{n} (a_i b_i - c_i) Y^{-i} = 0$$

The Q linear constraints are compressed,

$$\sum_{q=1}^{Q} (\mathbf{a} \cdot \mathbf{u}_q + \mathbf{b} \cdot \mathbf{v}_q + \mathbf{c} \cdot \mathbf{w}_q - k_q) Y^{q+n} = 0$$

Define polynomials

$$u_i(Y) = \sum_{q=1}^{Q} Y^{q+n} u_{q,i} \qquad v_i(Y) = \sum_{q=1}^{Q} Y^{q+n} v_{q,i}$$

$$w_i(Y) = -Y^i - Y^{-i} + \sum_{q=1}^{Q} Y^{q+n} w_{q,i} \qquad k(Y) = \sum_{q=1}^{Q} Y^{q+n} k_q$$

Combining the multiplicative and linear constraints,

$$\mathbf{a} \cdot \mathbf{u}(Y) + \mathbf{b} \cdot \mathbf{v}(Y) + \mathbf{c} \cdot \mathbf{w}(Y) + \sum_{i=1}^{n} a_i b_i (Y^i + Y^{-i}) - k(Y) = 0 \tag{15}$$

Sublinear Decision algorithm and WCP. During the first round of $\mathsf{AHP}_{\mathsf{Sonic}}$, the prover sends $r(X) = \sum_{-D}^{n} r_i X^i$ from which witness vectors $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ such that $a_i := r_i$, $b_i := r_{-i}$ and $c_i := r_{-n-i}$ for every $i \in [n]$ can be extracted.

Note that in the Sonic AHP decision algorithm, the only step that needs $O(|x|)$ computation is in computing the instance polynomial $k(Y) = \sum_{q=1}^{Q} Y^{q+n} k_q$ in order to evaluate it at y.

In our compiled argument, this is done by the prover and the verifier only checks a rational proof of evaluation given oracle access to the coefficient vector of $k(Y)$.

Protocol 2: $\mathsf{AHP}_{\mathsf{PLONK}}$

Offline phase. The indexer $\mathcal{I}$ receives as input $\mathbb{F} \in \mathcal{F}$ and $\mathsf{i} = (\mathbb{F}, n, m, l, \mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_O, \mathbf{q}_M, \mathbf{q}_C, \sigma, \mathcal{T}_\mathcal{C})$, and computes the following polynomial oracles as described before: selector polynomials $(q_L, q_R, q_O, q_M, q_C)$; preprocessed polynomials for permutation argument $(S_{L,\mathsf{ID}}, S_{R,\mathsf{ID}}, S_{O,\mathsf{ID}}, S_{L,\sigma}, S_{R,\sigma}, S_{O,\sigma})$; vanishing polynomial of $\mathbb{H}$, $v_\mathbb{H}(X) = X^n - 1$.

$$S_{L,\mathsf{ID}} = \sum_{i\in[n]} i \cdot L_i(X) \qquad S_{L,\sigma} = \sum_{i\in[n]} \sigma(i) \cdot L_i(X)$$
$$S_{R,\mathsf{ID}} = \sum_{i\in[n]} (n+i) \cdot L_i(X) \qquad S_{R,\sigma} = \sum_{i\in[n]} \sigma(n+i) \cdot L_i(X)$$
$$S_{O,\mathsf{ID}} = \sum_{i\in[n]} (2n+i) \cdot L_i(X) \qquad S_{O,\sigma} = \sum_{i\in[n]} \sigma(2n+i) \cdot L_i(X)$$

Input. $\mathcal{P}$ receives $(\mathbb{F}, \mathsf{i}, (\mathsf{w}_i)_{i\in[l]}, (\mathsf{w}_i)_{i\in[l+1,3n]})$ and $\mathcal{V}$ receives $(\mathbb{F}, (\mathsf{w}_i)_{i\in[l]})$ and oracle access to the polynomials output by $\mathcal{I}(\mathbb{F}, \mathsf{i})$.

Online phase: round 1. $\mathcal{P}$ computes $f_{\mathsf{pub}}(X), f_L(X), f_R(X), f_O(X)$ as described below and sends $(f_L(X), f_R(X), f_O(X))$ to $\mathcal{V}$.

$$f_L(X) = \sum_{i\in[n]} \mathsf{w}_i L_i(X) \quad f_R(X) = \sum_{i\in[n]} \mathsf{w}_{n+i} L_i(X) \quad f_O(X) = \sum_{i\in[n]} \mathsf{w}_{2n+i} L_i(X) \tag{8}$$

Online phase: round 2. On receiving challenges $\beta, \gamma \in \mathbb{F}$ from the $\mathcal{V}$, $\mathcal{P}$ computes $h_{\mathsf{ID}}(X)$, $h_\sigma(X)$ and a permutation polynomial $s(X)$ as described below.

$$h_{L,\mathsf{ID}} = f_L + \beta \cdot S_{L,\mathsf{ID}} + \gamma \qquad h_{L,\sigma} = f_L + \beta \cdot S_{L,\sigma} + \gamma \tag{9}$$
$$h_{R,\mathsf{ID}} = f_R + \beta \cdot S_{R,\mathsf{ID}} + \gamma \qquad h_{R,\sigma} = f_R + \beta \cdot S_{R,\sigma} + \gamma \tag{10}$$
$$h_{O,\mathsf{ID}} = f_O + \beta \cdot S_{O,\mathsf{ID}} + \gamma \qquad h_{O,\sigma} = f_O + \beta \cdot S_{O,\sigma} + \gamma \tag{11}$$
$$h_{\mathsf{ID}} = h_{L,\mathsf{ID}} \cdot h_{R,\mathsf{ID}} \cdot h_{O,\mathsf{ID}} \qquad h_\sigma = h_{L,\sigma} \cdot h_{R,\sigma} \cdot h_{O,\sigma} \tag{12}$$

$$s(X) = L_1(X) + \sum_{i\in[2,n]} \left(L_i(X) \cdot \prod_{1\le j<i} \frac{h_{\mathsf{ID}}(\zeta^j)}{h_\sigma(\zeta^j)} \right). \tag{13}$$

Then $\mathcal{P}$ sends an oracle polynomial $s(X)$ to $\mathcal{V}$.

Online phase: round 3. On receiving challenge $\alpha \in \mathbb{F}$ from the $\mathcal{V}$, $\mathcal{P}$ computes

$$\begin{aligned} F_\mathcal{C}(X) &= q_L(X) f_L(X) + q_R(X) f_R(X) + q_O(X) f_O(X) \\ &\quad + q_M(X) f_L(X) f_R(X) + q_C(X) + f_{\mathsf{pub}}(X) \\ F_1(X) &= h_{\mathsf{ID}}(X) s(X) - h_\sigma(X) s(\zeta X) \\ F_2(X) &= L_1(X)(s(X) - 1) \\ T(X) &= \frac{F_\mathcal{C}(X) + F_1(X) \cdot \alpha + F_2(X) \cdot \alpha^2}{v_\mathbb{H}(X)} \end{aligned}$$

and sends an oracle polynomial $T(X)$ to $\mathcal{V}$.

Query phase. $\mathcal{V}$ queries online oracles $(f_L(X), f_R(X), f_O(X), s(X), T(X))$ and all offline oracles with a random query point $z \in \mathbb{F}$. Moreover, it makes an additional query to the permutation polynomial $s(X)$ with ζz.

Decision phase. $\mathcal{V}$ first computes $f_{\mathsf{pub}}(X)$ as described: $f_{\mathsf{pub}}(X) = \sum_{i\in[l]} -w_i L_i(X)$. Then $\mathcal{V}$ computes $F_\mathcal{C}(z)$ (as in (7)), $F_1(z)$ and $F_2(z)$ based on the outputs of polynomial oracles. It then checks that $(F_\mathcal{C}(z) + F_1(z) \cdot \alpha + F_2(z) \cdot \alpha^2) = T(z) \cdot v_\mathbb{H}(z)$.

Figure 7: AHP for $\mathcal{R}'_{\mathsf{PLONK}}$

Protocol 3: $\mathsf{AHP}_{\mathsf{Sonic}}$

Offline phase. The indexer $\mathcal{I}$ receives as input $\mathbb{F} \in \mathcal{F}$ and $\mathsf{i} = (\mathbb{F}, n, Q, (\mathbf{u}_q)_{q\in[Q]}, (\mathbf{v}_q)_{q\in[Q]}, (\mathbf{w}_q)_{q\in[Q]})$, and computes the polynomial oracle $s(X,Y)$ as described below.

$$r(X,Y) = \sum_{i=1}^{n}(a_i X^i Y^i + b_i X^{-i} Y^{-i} + c_i X^{-n-i} Y^{-n-i}) \tag{16}$$

$$s(X,Y) = \sum_{i=1}^{n}\left(u_i(Y)X^{-i} + v_i(Y)X^{i} + w_i(Y)X^{i+n}\right) \tag{17}$$

$$t(X,Y) = r(X,1)(r(X,Y) + s(X,Y)) - k(Y) \tag{18}$$

Input. $\mathcal{P}$ receives $(\mathbb{F}, \mathsf{i}, (k_q)_{q\in[Q]}, (\mathbf{a},\mathbf{b},\mathbf{c}))$ and $\mathcal{V}$ receives $(\mathbb{F}, (k_q)_{q\in[Q]})$ and oracle access to the polynomials output by $\mathcal{I}(\mathbb{F}, \mathsf{i})$.

Online phase: first round. $\mathcal{P}$ computes $r(X,Y)$ and $t(X,Y)$ as described in Eq. (18). Mask $r(X,Y)$ as $r(X,Y) := r(X,Y) + \sum_{i=1}^{4} c_{n+i} X^{-2n-i} Y^{-2n-i}$ with random $c_{n+i} \in \mathbb{F}$ and send an oracle polynomial $r(X,1)$ to $\mathcal{V}$.

Online phase: second round. Upon receiving challenges $y \in \mathbb{F}$ from the $\mathcal{V}$, $\mathcal{P}$ sends an oracle polynomial $t(X,y)$ to $\mathcal{V}$.

Query phase. $\mathcal{V}$ queries online oracles $r(X,1)$ and $t(X,y)$ with a random query point $z \in \mathbb{F}$. Moreover, it makes additional queries to $r(X,1)$ with yz and to $s(X,Y)$ with (z,y).

Decision phase. $\mathcal{V}$ first computes an instance polynomial $k(Y)$ as described in the text. Then $\mathcal{V}$ checks that

$$t(z,y) = r(z,1)(r(yz,1) + s(z,y)) - k(y).$$

Figure 8: AHP for $\mathcal{R}_{\mathsf{Sonic}}$