A Theory of Composition for Proofs of Knowledge

Abhiram Kothapalli

CMU-CS-24-126

May 2024

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Bryan Parno, Chair
Aayush Jain
Elaine Shi
Srinath Setty (Microsoft Research)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was supported by a fellowship from Protocol Labs, a gift from Bosch, NSF Grant No. 1801369 and 190099, and by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Proofs of Knowledge, Composition, Recursion

For friends, family, and the few in between

Abstract

In 1985, Goldwasser, Micali, and Rackoff introduced a compelling new notion of a proof, known as a *proof of knowledge*, in which a verifier interactively checks that a prover knows a satisfying witness for a claimed mathematical statement. For example, a verifier may check that a prover knows a witness solution for a prescribed Sudoku problem (and more generally any problem in NP). Interestingly, interaction affords seemingly paradoxical properties. For instance a prover can interactively demonstrate knowledge of a witness without revealing any information about it. Moreover, the demonstration itself can can be significantly shorter than the witness. For the past four decades, we have made significant progress in theoretical computer science by studying the time complexity, space complexity, and communication complexity of these interactions. Remarkably, we are also seeing proofs of knowledge being used in cryptographic applications to secure *billions* of dollars worth of assets.

Today, however, in search of practical efficiency, a growing body of work challenges the traditional paradigm by describing interactions in which the verifier does not fully resolve the prover's statement to true or false but rather reduces it to a simpler statement to be checked. Such interactive reductions, although central to modern proofs of knowledge, lack a unifying theoretical foundation. Towards a common language, we introduce *reductions of knowledge*, which reduce the task of checking knowledge of a witness in one relation to the task of checking knowledge of a witness in another (simpler) relation. We show that reductions of knowledge can be composed naturally, and thus serve as both a unifying abstraction and a theory of composition. As such, reductions of knowledge formalize a simple, but subtly powerful new perspective that *proofs of knowledge are maps between propositions of knowledge.*

We demonstrate that this is not merely a theoretical insight. In a very tangible sense, this perspective is becoming increasingly important for developing modern proofs of knowledge. We show that large swathes of the extant literature can be expressed crisply in our framework, as well as develop new techniques that cannot be expressed in preexisting models. Indeed, one of the early successes of our framework is the introduction of *folding schemes*, which are a particular type of two-to-one reduction of knowledge. We show that folding schemes have become an important tool for developing *recursive* proofs of knowledge, that is, proofs that demonstrate knowledge of other proofs. Recursive proofs have the potential to significantly scale decentralized computation due to their unique ability to handle stateful computation with dynamic control flow. As a result, recursive proofs (and the underlying folding schemes) have received significant recent interest in both industry and academia.

Acknowledgments

First and foremost, I would like to thank my committee, Bryan Parno, Aayush Jain, Elaine Shi, and Srinath Setty, who through years of guidance, have helped shape this thesis into what it is today. I would like to thank my advisor, Bryan, for tirelessly working to bring out the hidden potential he saw in me six years ago. The central direction of this thesis is inspired by Bryan's own research program, which masterfully blends the best ideas from opposite ends of computer science. I would like to thank Aayush Jain for his meticulous attention to detail to the technical aspects of this work. I would like thank Elaine Shi for being one of the first to see the potential of the ideas in this thesis, and offering sage technical advice that has greatly improved central aspects of this work. Finally, I would like to thank Srinath for helping set the central technical applications of this thesis; a large portion of this thesis is the direct result of years of collaboration, discussion, and debate over every last technical detail.

I would like to thank the Secure Foundations Lab, which over the years has included Yi Cai, Chanhee Cho, Samvid Dharanikota, Aymeric Fromherz, Sydney Gibson, Travis Hance, Benjamin Lim, Zhengyao Lin, Steve Matsumoto, Mike McLoughlin, Pratap Singh, Xueyuan Zhao, and Yi Zhou. I would like to especially thank Jay Bosamiya and Josh Gancher for their significant mentorship in both academic and personal matters. I would also like to thank Lisa Masserova for growing with me every step along the way from a first year graduate student to a working cryptographer.

I would like to thank the larger academic community at Carnegie Mellon University, which blurs the distinction between computer science, mathematics, and philosophy. I strongly believe this thesis would not have manifested in any other environment. I would like to especially thank the attendees of the category theory reading group, the principles of programming seminar, the theory seminar, and the cryptography seminar. The best ideas always came from the tension between their individual wisdoms and dogmas.

Finally, I would like to thank my friends, my family, and the few in between that blur the distinction. This work is for you.

Abhiram Kothapalli
Pittsburgh, Pennslyvania
May 2024

Contents

Chapter 0

Conspectus

We deal with those proofs that can be "explained in class". In a classroom, the lecturer can take full advantage of the possibility of interacting with the "recipients" of the proof.

– Shafi Goldwasser, Silvio Micali, and Charles Rackoff,
The Knowledge Complexity of Interactive Proof-Systems

What is a proof? Commonly, we understand a proof as irrefutable evidence of the validity of a statement. More pedantically, we can understand proofs as deriving the validity of a statement (in a logical manner) from a set axioms which we believe are irrefutably true.

Perhaps the earliest modern manifestation of this interpretation dates back to 300 BC in Euclid's *Elements* which postulates the following axioms for a geometric system (now known as Euclidian geometry).

(1) There exists a straight line between any two points.

(2) A straight line can be extended indefinitely.

(3) A circle is precisely determined by a point and a radius.

(4) All right angles are equal to one another.

(5) If a straight line intersecting two straight lines forms acute angles on the same side, the two straight lines eventually intersect on that side with the acute angles.

Euclid then carefully derived consequence after consequence by meticulous application of these axioms and preceding consequences. For instance, in Book I, Proposition 32 of *Elements*, Euclid derives the following.

The sum of the angles of any triangle will always equal 180 degrees.

In the last millennia, this approach of stating axioms, deriving consequences, and then deriving more interesting consequences has set the standard for mathematical inquiry.

Indeed, this process has been so successful, that it is still reenacted today with clinical precision in the standard American middle school geometry class.

Euclid's proofs themselves proceed by implicitly invoking basic *logical* axioms in addition to those pertaining to the geometric system. For example, consider the following logical axioms.

- If A is true and B is true then $A \wedge B$ is true.
- If $A \wedge B$ is true then A is true.
- If $A \to B$ is true and A is true then B is true.

Such axioms define *propositional logic*, that is, the calculus of manipulating true/false assertions. Of course, it is our responsibility, as a reader, to assign semantic meaning to the axioms of propositional logic. For instance, we interpret $A \wedge B$ to mean "A and B" and $A \to B$ to mean *"if A is true then B is true"*. Much like we can derive interesting geometric consequences from Euclid's axioms, we can derive interesting logical consequences from the axioms of propositional logic. For instance, we can derive the following consequences.

- If $A \to B$ and $B \to C$ then $A \to C$
- If $A \to B$ and $C \to D$ then $A \wedge B \to C \wedge D$.

In the context of Euclidian geometry, we can view the terms such as A and B as being directly populated with the axioms or subsequently derived consequences of the geometric system. We can then view the logical system as the engine that transforms these consequences into new, and more interesting consequences.

Propositional logic, and other logical systems give us a means to interpret proofs themselves as mathematical objects with precise rules of manipulation. In the early 20th century, led by the likes of L.E.J Brower, Arend Heyting, and Andrey Kolmogorov, mathematics imbued such proofs with computational semantics: Proofs themselves became *programs* that constructed witnesses for the postconditions given as input witnesses to the preconditions. These witnesses are either assumed axiomatically, or constructed from prior proofs. In this regime, the proposition $A \to B$ semantically means that *"there exists a program which given a witness for the proposition A, constructs a witness for the proposition B"*. A valid program that satisfies this desiderata itself forms a witness for the proposition $A \to B$, which in turn can be used as input for proofs for *higher-order* propositions such as $(A \to B) \wedge (B \to C) \to (A \to C)$.

This *proofs as maps* paradigm, formally referred to as constructive logic [88], is further cemented by the celebrated Curry-Howard correspondence, which observes a symmetry between the structure of proofs and the structure of programs [89]. This correspondence eventually inspired Martin-Löf's *type theory* [110] which interpreted programs as proofs of a specification. The conflated view of proofs and programs is central to our modern interpretation of logic, type theory, programming languages, and formal methods. Today, constructive logic has enabled formal verification tools for demonstrating that a program behaves precisely as specified, a nonnegotiable requirement in safety-critical systems ranging from internet banking to space exploration. This paradigm has additionally enabled

computer-aided verification of mathematical proofs, opening up a path to mechanizing large swathes of mathematics.

While Brower and his contemporaries proselytized constructive logic in Europe, Kurt Gödel settled down in the Americas, and began to study the limits of what we could and could not prove efficiently under known logical systems. In a letter to John von Neumann, Gödel states the following.

> One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n. Let $\psi(F, n)$ be the number of steps the machine requires for this and let $\varphi(n) = \max_F \psi(F, n)$. The question is how fast $\varphi(n)$ grows for an optimal machine. One can show that $\varphi(n) \geq k \cdot n$. If there really were a machine with $\varphi(n) \sim k \cdot n$, this would have consequences of the greatest importance.

Gödels musings were eventually formalized by Stephan Cook and became the most important question in theoretical computer science: *Suppose a proposition, if true, will have a short proof; does this mean it is easy to prove?* This can be stated more precisely in the language of computational complexity theory: *Is the complexity class* NP *equivalent to the complexity class* P*?*

As a concrete example, let us define graphs as a collection of vertices and a collection of edges between these vertices. Given an arbitrary graph G, can we, using just three colors, color every vertex such that no two connected vertices have the same color. This problem, known as *Graph three-colorability*, is naturally equivalent to finding a proof for propositions of the form "Graph G has a three-coloring" More interestingly, we know that this problem is in NP: Given a coloring, we can efficiently enumerate each edge and check that it has a different color on each end. We do not know, however, if there is an efficient algorithm to find this coloring in the first place. If there is, because we can show that any proposition in NP can be encoded as a graph three-coloring proposition, we will be able to demonstrate that $\mathsf{P} = \mathsf{NP}$. As Gödel remarks, *this would have consequences of the greatest importance*. Indeed, in the same letter, Gödel later states that *"[If* $\mathsf{P} = \mathsf{NP}$*], the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine."*

With such monumental stakes, in the late 20th century, the most interesting questions about the nature of a proof became less about the *structure* of a proof and more about the *power* of a proof. The exciting new field of computational complexity theory began to ask and answer promising questions such as

- Is writing proofs for propositions in class A (e.g., graph three-coloring) equally as hard as as writing proofs for propositions in class B (e.g., boolean constraint satisfiability)?
- How long must proofs be for certain classes of propositions? What if we allowed for some amount of soundness error?

- How much space (i.e., working memory) would it take to generate proofs for a certain classes of propositions?

- Are there certain proof systems which afford significantly shorter proofs?

It was in this climate that in 1985, Shafi Goldwasser, Silvio Micali, and Charles Rackoff asked a seemingly simple question: What if we allow for interaction?

> We introduce interactive proof-systems to capture a more general way of communicating a proof. We deal with those proofs that can be "explained in class". Informally, in a classroom, the lecturer can take full advantage of the possibility of interacting with the "recipients" of the proof. They may ask questions at crucial points of the argument and receive answers.

This inquiry eventually won them a Turing award, and launched a research program that, after four decades, is more active than ever. Interaction, it seems, affords paradoxical properties that update our understanding of what constitutes a valid proof. For instance, a prover can interactively convey just enough information to demonstrate that it knows a witness without revealing any information about this witness. Interactions that satisfy this desiderata are referred to as *zero-knowledge* proofs. Alternatively, the length of the interaction (i.e., the total size of all messages sent) can be orders of magnitude smaller than the witness itself. Naturally, these interactive proofs must be probabilistic in nature, as the prover always has a slight chance of guessing the right responses to the verifier's questioning even if it does not know a witness. However, this probabilistic relaxation is essential to the potency of interactive proofs: A proof of an proposition no longer needs to be irrefutable evidence, but rather any object that would be extraordinarily difficult to produce if the proposition was not true.

More formally, Goldwasser, Micali, and Rackoff refer to their interpretation of a proof as a *proof of knowledge* (alternatively *argument* of knowledge). A proof of knowledge is an interactive protocol in which a prover demonstrates to a verifier that it knows a witness for some claimed computational statement. Of central importance is that the prover demonstrates that it *knows* a witness rather than demonstrating that there *exists* a witness. For example, demonstrating knowledge of a prime factorization for a large composite number is qualitatively different from demonstrating that there exists one (which is always true). Capturing this subtlety formally is part of the challenge and elegance of their notion. To distinguish propositions of this form, we refer to them as *propositions of knowledge*.

As a concrete example, we present a *zero-knowledge* interaction for proving graph three-coloring propositions. Suppose the prover knows a valid coloring C (the witness) for a graph G (the proposition). She convinces a verifier that it knows this coloring without revealing it as follows.

1. The prover randomly permutes all the colors. For example, red vertices are relabeled as blue vertices, blue vertices are relabeled as red vertices, and green vertices stay fixed.

2. The prover sends *hiding commitments* to the colors for each vertex. In practice, this commitment can be a SHA256 hash of the color and some fresh randomness.

3. The verifier picks a random edge and asks the prover to open the commitments associated with the corresponding endpoints. In practice, the verifier asks the prover to send openings (i.e., preimages) to the pair of SHA256 hashes.

4. The verifier checks that the openings indeed refer to different colors.

5. The verifier repeats starting at Step 1 until it is sufficiently convinced.

The intuition of the above proof is as follows: If the prover does not actually know a valid coloring, then there must exist at least one edge the prover commits to where the color of both endpoints is the same. For N edges, the verifier only has a $1/N$ chance of picking this edge, but it is free to test as many times as it likes. For instance, if it repeats the interaction for k times, then the prover only has a $(1 - 1/N)^k$ of deceiving the verifier.

Of course, this only works because a hiding commitment ensures that the prover cannot change her answer after the verifier has picked out an edge. Moreover, because the commitments reveal no information about their openings, the verifier will only learn the colors of the two vertices it requests in each round. However, because the prover rerandomizes the coloring in each round, these openings are not compatible with each other, and thus cannot be stitched together to reveal the full coloring. In fact, we can show that the full interaction transcript is effectively independent of the witness (which is why it reveals no information). However, to the verifier, this is just as good of a proof as any.

As stated earlier, because any proposition in NP can be stated as a graph three-coloring proposition, this interaction applies to any proposition in NP. That is, *any proposition that is easy to verify, can be easily verified in zero-knowledge.*

For the past four decades, this strange new notion of a proof has enabled celebrated results in computational complexity theory. For instance, if the verifier is allowed randomness (as in the above zero-knowledge proof for graph three-coloring), then it can efficiently verify any proposition that takes even a polynomial amount of *space* to prove ($\mathsf{IP} = \mathsf{PSPACE}$). Alternatively, if the verifier is allowed to challenge two non-colluding provers, then it can efficiently verify any proposition that takes a non-deterministic algorithm exponential time to prove ($\mathsf{MIP} = \mathsf{NEXP}$). More recently, an exciting new result demonstrated that if these provers shared quantum entanglement, then they can efficiently demonstrate any class of propositions that is *recursively enumerable* ($\mathsf{MIP}^* = \mathsf{RE}$). This includes any class of propositions where there exists an algorithm to enumerate all true propositions (a mind-boggling number of classes fit this description).

While such results are largely theoretical in nature, this is no longer the case for proofs of knowledge as a whole. Today, industrial-grade applications demand proofs of knowledge that are zero-knowledge, succinct (the proof is significantly shorter than the witness), and non-interactive (i.e., the prover sends a single message consisting of a simulated interaction transcript). Zero-knowledge, succinct, non-interactive proofs of knowledge, or zkSNARKs for short, are being used in practice today to secure *billions* of dollars worth of assets. Modern zkSNARKs enable a new class of secure applications with enhanced integrity and

privacy guarantees such as verifiable databases, private voting protocols, anonymous credentials, and private cryptocurrencies. As such zkSNARKs have become a hotly studied topic in applied cryptography, computer security, and privacy research.

Unfortunately, in all the exciting developments in the late 20th century, we seemed to have distorted our original understanding of a proof: The supposedly fundamental discovery that "proofs are maps" shows up nowhere in the theory of interactive proofs. Thus, at the turn of the 21st century, we were left with two seemingly unrelated, but independently successful, interpretations of a proof: Computational complexity theory tells us that "interactions are proofs" while constructive logic tells us that "proofs are maps". Both interpretations have fostered remarkably fertile (and ongoing) research programs which bear no resemblance to each other.

That is, until about five years ago, in the search of practical efficiency, a growing body of work began to describe interactions in which the verifier does not fully resolve the prover's claim to true or false, but rather reduces it to a simpler statement to be checked. This unknowingly opened up a path to interpreting such interactions as maps: An interaction which enables a verifier to reduce the task of checking a proposition of knowledge A to the task of checking a (simpler) proposition of knowledge B is a witness for the proposition $A \to B$. This can be summarized as a simple, but subtly powerful new perspective that bridges the two paradigms:

Proofs of knowledge are maps between propositions of knowledge.

The goal of this thesis is to capture this statement with mathematical precision and then demonstrate the flavor of insights that it affords. A secondary goal is to demonstrate that this is not merely a theoretical insight. In a very tangible sense, this perspective is becoming increasingly important for developing state-of-the-art zkSNARKs being used in practice today, some of which were introduced in the works comprising this thesis.

The remainder of the thesis serves as a detailed mathematical portrait of what has already been said in words. We start by redefining *proofs of knowledge* in a way that helps us naturally interpret them as maps over *propositions of knowledge*. Next, we demonstrate that proofs of knowledge indeed behave like *maps* in the sense that they support functional application and composition. At this point, we will have nominally demonstrated the thesis statement.

However, the real burden lies in demonstrating the more substantive claim that *this* is how we should interpret proofs of knowledge moving forward. Here, we make our point by fitting large swathes of the extant literature crisply within our model, as well as developing new techniques that cannot be expressed in preexisting models. Indeed, one of the early successes of our formalization of the proofs-as-maps perspective is the introduction of *folding schemes*, which efficiently reduce the task of checking two propositions of the same form into the task of checking a single proposition of the same form. We demonstrate that it is possible to fold propositions expressive enough to capture any statement in NP. We then show that such folding schemes can be used to develop highly-efficient *recursive proof systems*, that is, proofs that demonstrate knowledge of other proofs. Recursive proofs have the potential to significantly scale decentralized computation due to their unique ability

to handle stateful computation with dynamic control flow. As a result, recursive proof systems (and the underlying folding schemes) have received significant recent interest in both industry and academia.

Putting everything together, and squinting through the lens of *category theory* (the general theory of composition) we derive broad insights into the structure of proofs of knowledge as a whole. In many ways they will mimic the structure of proofs in propositional logic that we have known for centuries. And in many ways they will not. It is in this tension, we hope, that one will find another valuable glimpse into the nature of a proof.

Chapter 1

Introduction

Mathematics is the art of giving the same name to different things ... when language has been well chosen, one is astonished to find that all demonstrations made for a known object apply immediately to many new objects.

– Henri Poincaré,
The Future of Mathematics

If the first generation of modern cryptography is about guaranteeing the integrity and privacy of *data*, then the second generation is certainly about guaranteeing the integrity and privacy of *computation*. This is perhaps most emphatically demonstrated by the meteoric rise of *zero-knowledge proofs of knowledge*, a powerful technique for proving the

correctness of computations without revealing any secret inputs. In this chapter, we review proofs of knowledge and overview the central contribution of this work, which is a *theory of composition* for proofs of knowledge.

1.1 Overview of Interactive Proof Theory

As introduced in Chapter 0, zero-knowledge proofs of knowledge are short certificates that attest to the correct execution of a computation without revealing any secret inputs. Today, zero-knowledge proofs are being used to secure *billions* of dollars worth of assets [28, 122]. Zero-knowledge proofs enable a new class of secure applications with enhanced integrity and privacy guarantees such as verifiable databases [15, 140, 141, 142], private voting protocols [144], anonymous credentials [63, 76], private cryptocurrencies [28, 61, 122], and verifiable virtual machine execution [14, 27].

These impressive achievements stem from the interpretation set forth by proofs of knowledge: A "proof" is any object that is infeasibly difficult to produce if the proposition is not true. This seemingly simple relaxation affords properties such as zero-knowledge and succinctness, which are the crux of why proofs of knowledge are so attractive in practice.

However, there is a vast gulf between the originally introduced proof systems for, say, graph three-coloring and the proof systems of today for industrial applications such as zero-knowledge virtual machines. In this section, we discuss key results and techniques that paved the path from theory to practice.

1.1.1 Inventing Proofs of Knowledge

The central contribution of Goldwasser, Micali and Rackoff [83] is the notion of an *interactive* proof and the notion of *zero-knowledge*. A proof of knowledge (alternatively argument of knowledge) is an interactive protocol in which a *prover* proves a *proposition of knowledge* to a *verifier* (e.g., I know a secret satisfying assignment x for boolean formula φ)

Intuitively, zero-knowledge means that the interaction transcript reveals no information about the witness to the proposition of knowledge. Goldwasser et al. judiciously define zero-knowledge around *computational* equivalence. Two distributions are said to be computationally equivalent if a probabilistic polynomial time *distinguisher* (i.e., a randomized algorithm that can only take a polynomial number of steps in the size of its input) cannot correctly identify the provenance of a value that is randomly picked from one of these two distributions. A proof of knowledge, then, is zero-knowledge if there exists a probabilistic polynomial time *simulator* which can produce an interaction transcript that is computationally equivalent to that produced by an honest prover-verifier interaction. Central to this notion is that the simulator is not provided the witness, and thus its ability to produce such a transcript asserts its independence from the prover's witness.

The definition of zero-knowledge can be strengthened to *statistical* zero-knowledge, where the distinguisher is given unlimited computational capabilities. Alternatively, zero-knowledge can be weakened to alternate notions, such as *witness-indistinguishability*, where a proof does not reveal *which* witness it is associated with among a predefined set of

satisfying witnesses. Studying the permissible set of proof systems under such nuanced strengthenings and weakenings is a recurring theme in the literature.

Recall from Chapter 0 that Goldwasser et al. distinguish interactions that demonstrate *knowledge* of a witness from interactions that merely demonstrate the *existence* of a witness. The precise definition of *knowledge* is hinted at, but not actually provided in the original work. Bellare and Goldreich [20] provide the canonical definition by summarizing and improving upon prior attempts [66, 68, 131]: A proof of knowledge is said to be *knowledge-sound* if any expected polynomial-time prover that can produce accepting proofs induces a corresponding expected polynomial-time *extractor* that can retrieve the underlying witness given access to the "source code" of the prover. This elegant definition, allows us to understand knowledge of a value as the "ability to efficiently compute" this value.

As with zero-knowledge, the definition of knowledge soundness can be strengthened or weakened to interesting effect. For instance, Pass [118] introduces the notion of *straight-line* extractability, which only permits extractors that have a constant runtime overhead over the prover. As we show, interactions that satisfy this property can be composed arbitrarily. Alternatively, Lindell [104] introduces *witness-extended* emulation, where the extractor is additionally responsible for simulating a transcript that agrees with the extracted witness. Such a definition makes it easier to prove the soundness of larger cryptographic protocols using proofs of knowledge as a subroutine.

1.1.2 The Arithmetization Revolution

Even as definitions ossified in the early 1990s, we were still no closer to designing practical proof systems: Proof systems still targeted problems such as graph three-coloring, which were not practical languages to encode propositions of knowledge that one might encounter in practice. Moreover, while the proofs for these languages afforded properties like zero-knowledge, they still scaled (sometimes egregiously so) with the size of the original witness.

Although largely celebrated as a landmark theoretical result, the first step towards practicality came from the $\mathsf{IP} = \mathsf{PSPACE}$ result by Adi Shamir [128]. Shamir demonstrated that any proposition that can take up to polynomial *space* to prove still has an interactive proof that takes only a polynomial amount of *time* to verify. Keep in mind that polynomial-space is vastly more permissible than polynomial-time. To build a proof system that admitted such a significant discrepancy between the power of the prover and that of the verifier, Shamir invented a new technique called *arithmetization*. As Aaronson and Wigderson [10] put it *"the idea was that, instead of treating a Boolean formula φ as just a black box mapping inputs to outputs, one can take advantage of the structure of φ, by 'promoting' its AND, OR, or NOT gates to arithmetic operations over some larger field $\mathbb{F}$. One can thereby extend φ to a low-degree polynomial $\widetilde{\varphi}$"*. While every constraint of the original formula φ must be manually checked, the polynomial $\widetilde{\varphi}$ has useful *error-correcting* properties. This permits the verifier to query only a handful of random points to convince himself with near certainty that the prover knows a valid witness.

In particular, Shamir equates the task of checking the original formula φ to the task of checking that the sum of evaluations of the corresponding multivariate polynomial $\widetilde{\varphi}$ on

the hypercube is some value σ. This *sum-check* can be expressed as follows.

$$\sigma = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} \widetilde{\varphi}(x_1, \ldots, x_n). \tag{1.1}$$

Checking this equation manually would require checking an *exponential* number evaluations of $\widetilde{\varphi}$ (in the number of variables). However, Shamir then invokes an equally famous result due to Lund et al. [106], which shows that, using interaction, the verifier can reduce the task of checking Equation 1.1 to the task of checking just a single point

$$\sigma' = \widetilde{\varphi}(r_1, \ldots, r_n)$$

with only *logarithmic* time and communication complexity. Here, random points $r_1, \ldots, r_n \in \mathbb{F}$ and updated claim $\sigma' \in \mathbb{F}$ are produced during interaction. Shamir then proceeds to show that this final check can be done in polynomial time.

Thus, Shamir leveraged the sumcheck protocol as an Archimedian lever with which a verifier can command a vastly more powerful prover. This has since cemented the sumcheck protocol as the central tool for designing proofs of knowledge. More broadly, Shamir's general strategy of arithmetization guides virtually every modern proof system. Generally, proof systems today are achieved by putting together the following pieces:

1. a constraint system (e.g., a system of quadratic equations) to represent computational statements as low-level algebraic constraints,
2. mathematical representations (e.g., polynomials) to arithmetize constraints and purported satisfying assignments,
3. efficient algebraic tests (e.g., polynomial equality testing) to check that the encoded assignment satisfies the prescribed algebraic constraints, and
4. cryptographic machinery (e.g., hiding commitments) to prove succinctly and in zero-knowledge that the prescribed algebraic tests are satisfied.

This strategy gave a general platform to swap pieces in and out as we discovered newer, better techniques. Additionally, because this strategy demonstrated how to reduce general constraint satisfiability to simpler *algebraic* questions, we began to refocus our attention on designing efficient proof systems for algebraic propositions of knowledge. Proof systems for the following algebraic propositions are especially popular.

- Knowledge of a polynomial under a commitment that evaluates to a claimed value at a prescribed point.
- Knowledge of witness vectors under commitments such that their inner-product produces a claimed value.
- Knowledge of a vector under a commitment that is in the rowspace of a public matrix.

As such, general proof systems for NP became general platforms that orchestrated a family of such algebraic proof systems to prove a complex proposition in NP.

1.1.3 Taming Complexity with Idealized Models

As general proof systems for NP grew in complexity, we began to rely on abstractions to help manage the complexity: Cryptographers designed models that abstracted out implementation details related to communication, randomness generation, and so on.

The most well-known is the *probabilistically checkable proofs* (PCP) model which idealizes the prover communication with the verifier. In particular, in the PCP setting, a computationally capable prover is allowed to write a large fixed proof "in the sky" (formally modeled as an oracle instantiated by the prover). A computationally limited verifier is then allowed to query this proof at a handful of points. The celebrated *PCP theorem* by Arora and Safra [13] demonstrates that the verifier only needs to read *three bits* for some non-trivial measure of soundness. Kilian [94] is the first to instantiate such PCPs in "real life", by replacing the oracle with a concrete commitment to the entire contents of the oracle that can be verifiably queried with efficient *lookup* proofs. In particular, this commitment is implemented using a *Merkle tree* [113], which enables the verifier to efficiently open this commitment to specific substrings.

The *interactive oracle proof* (IOP) model [26] is a popular relaxation of the PCP model, which much better abstracts the design patterns of modern proof systems. In essence, the IOP model allows the prover to commit to a proof oracles in each round of interaction, which the verifier can query at any subsequent point in the interaction. *Polynomial interactive oracle proofs* [44] (PIOP), as the name suggests, restrict the oracles to represent polynomials, thereby baking the arithmetization step into the model. These models are similarly instantiated using Kilian's approach of replacing oracles with efficiently queryable commitments.

Another popular model is the *random oracle model* [21] (ROM), which assumes an idealized (but reproducible) source of randomness that is accessible to all parties called the random oracle. Such an oracle reduces or, in many cases, completely removes the need for interaction (while still retaining succinctness) as the prover can verifiably sample the verifier's random challenges using the random oracle rather than querying it through interaction. Of course for such a proof of knowledge to be usable in practice, the random oracle, must be instantiated with a concrete primitive that we *heuristically* assume approximates idealized source of randomness. Traditionally, a cryptographic hash function, such as SHA256, is considered a suitable instantiation.

Typically, the only restriction on an adversarial prover's capabilities is that they run in (probabilistic) polynomial-time. This is sometimes too permissive. Idealized soundness models [71, 111, 129] capture more fine-grained assumptions about adversary's capabilities in practice, thereby enabling security proofs for systems against realistic adversarial strategies. For instance, the *algebraic group model* (AGM) [71] states that given a randomly sampled public key $g_1, \ldots, g_n \in \mathbb{G}$, whenever an adversary produces a value $c \in \mathbb{G}$ it must additionally offer an "explanation" $a_1, \ldots a_n$ such that $c = \prod_{i \in [n]} g_i^{a_i}$. In doing so, the AGM captures the assumption that an adversary must *know* the preimage to all the group elements it produces during interaction. Of course, this assumption is trivially undermined by an adversary that outputs, say, random group elements. Thus, security in the AGM is only a meaningful when such strategies are presumably not profitable.

Idealized models afford a setting for *provable* soundness guarantees for proof systems. However, instantiating these models typically requires heuristic assumptions. For instance, we assume that replacing the random oracle with SHA256 (in the case of the ROM), or producing group elements without a known preimage (in the case of the AGM), does not afford the adversary better odds at tricking the verifier. Such heuristic assumptions are typically non-falsifiable, in the sense that we cannot demonstrate a counterexample invalidating them. In some cases they are plainly false, as Goldwasser and Kalai [81] demonstrated with the random oracle assumption for specially contrived schemes. As a result, heuristic assumptions can only be taken on faith for any particular scheme and are actively contested, with some cryptographers opting to avoid them altogether. Unfortunately, Gentry and Wichs [79] demonstrate that succinct, non-interactive proof systems, must, in one way or another, rely on such heuristic assumptions. Thus, the best we can do is to try to rely on more plausible, and well-accepted heuristic assumptions.

1.1.4 Transformations over Proofs of Knowledge

Thus far, we have only discussed interactive proofs of knowledge. However, practical applications demand non-interactivity: For instance, a (zero-knowledge) proof that a Bitcoin transaction is valid should be a short certificate that can be publicly checked by all participants in the network rather than an invitation to participate in an interaction.

This gap is bridged by a cornerstone technique in interactive proof theory, the *Fiat-Shamir transformation*, which takes any interactive proof system where the verifier only sends random challenges, and converts it into a non-interactive proof system. The core idea is to have the prover honestly simulate the verifier's randomness using a random oracle (or a hash function), and then let the simulated interaction transcript stand in as a static proof of knowledge. This allows proof system designers to work in a more natural interactive setting, and then compile their designs into a version usable in practice. The Fiat-Shamir transformation is the reason why interactive proofs are conceptually understood as *"signatures of correct computation"* [116]. The subsequent Fischlin transformation [69] also achieves non-interactivity, but also achieves straight-line extractability along the way.

The *Cramer-Damgard* transformation [60] compiles a proof system where the prover only sends field elements into a zero-knowledge proof system. The core idea is to have the prover instead send hiding commitments to the field elements, and then prove that the field elements under these commitments satisfy the verifier's checks. If the verifier's checks are linear, such field elements can be blinded by taking a random linear combination with masking element which still ensures that the verifier's checks still pass. This technique has been successfully adapted in various settings [35, 53, 117], demonstrating that zero-knowledge is also a property that can be achieved external to the core design.

1.1.5 Towards Practical zkSNARKs

In the early 2010s, we had on hand a general arithmetization-based framework, generic transformations to achieve desirable properties, and idealized models to prove these properties in. Together, these gave us a powerful theoretical toolkit needed to start engineering

practical proofs of knowledge.

Applications such as cryptocurrencies, and anonymous certificates demanded proofs of knowledge that were **(1)** zero-knowledge, **(2)** succinct (i.e., short), and **(3)** non-interactive. Proofs of knowledge that satisfied these requirements were referred to as zero-knowledge succinct non-interactive arguments of knowledge, or *zkSNARKs* [29] for short. We were also largely interested in propositions of the form *"I know a secret x, such that function F outputs y, on input x"*. That is, propositions attested to the correct execution of a function (on secret inputs). Zero-knowledge SNARKs for such propositions enabled *verifiable computation* [77] which enables a weak client to outsource expensive computations to an untrusted (but powerful) server.

In 2008, Goldwasser, Kalai, and Rothblum [84] provided a proof system which would serve as a template a number of practical proof systems in the decade to come. In particular, they provide a proof system for circuit satisfiability, but instead of arithmetizing the entire circuit, they demonstrate that the prover and verifier can more efficiently by iteratively arithmetizing and checking each layer. Taking Shamir's approach, in each layer the prover and verifier utilize the sum-check protocol. While this reduces the verifier's time complexity, little is done to reduce the prover's time complexity. This is remedied by Cormode, Mitzenmacher, and Thaler [59], who demonstrate that the prover can participate in the sumcheck protocol with only $n \log n$ time complexity, where n the number of coefficients of the polynomial thus giving one of the first practical proof systems for NP.

Wahby et al. [135] demonstrate that the prover and verifier time complexity can be further improved for *log-space uniform* circuits (where the description of the circuit is asymptotically smaller than the circuit itself). Subsequently, Wahby et al. [136] achieve a zero-knowledge and a verifier with sublinear time complexity for data-parallel circuits using the Cramer-Damgard transformation. Further work [137, 139] demonstrates that the prover time complexity for the sumcheck protocol can be reduced to linear in the size of the polynomial.

In a parallel vein, in 2013, Gennero, Gentry, Parno, and Raykova (GGPR) [78] achieved a *constant-time* verifier and a nearly linear prover for general computations by making use of a per-circuit trusted setup (that is, a trusted party would have to create a public key to verify against). Core to their work is a new constraint system, quadratic arithmetic programs, which is the dominant method for efficiently representing computations today. Parno, Gentry, Howell, Raykova [117], introduced Pinocchio shortly afterwords, which implemented the GGPR protocol, making it the first implemented zkSNARK. This was closely followed by several more implementations [24, 122] of further optimized proof systems [86]. Finally, after nearly three decades, we had tangible proof systems that could be run on tangible computations.

The first implemented proof systems marked the beginning of a new era for proof system design in the 2010s. For the first time, we began to ask and answer *engineering* questions of the following flavor:

- Which applications should we start to optimize proof systems for?
- What sort of setup assumptions are acceptable to the average user?

- What concrete groups and security parameters (i.e., key length) can we use that maximizes performance while maintaining security?

- How can we design SNARK libraries in a way that is easy to maintain and update?

To address these questions there has been a long line of work (with multiple subbranches) working to make zkSNARKs **(a)** more practical and **(b)** with better cryptographic and setup assumptions.

The first commercially successful application of zkSNARKs was zCash [122], which was a cryptocurrency that preserved the privacy of the transactions (e.g., involved parties and total amount), using the GGPR proof system. ZCash also demonstrated the first growing pain of practical zkSNARKs: The seemingly benign trusted setup process of GGPR, magnified into trusting a single entity to honestly generate a public key to be used by millions of users. If this entity had generated the public key maliciously it could grant users the power to forge any transaction of their choosing. To partially circumvent this issue, zCash invented a complex "Powers-of-Tau" setup procedure, which distributed the key generation process to several presumably non-colluding parties. This kicked off one of the most prominent research programs in the late 2010s, which was to reduce trust in the setup process while maintaining (and even improving) performance.

One of the first contributions in service of this goal, was due to Bootle, Cerulli, Chaidos, Groth, and Petit (inspired by Bayer and Groth [19]), who discovered that inner-products can be efficiently proven without a trusted setup by recursively *reducing* the task of checking a size n inner-product to the task of checking a size $n/2$ proof. A flurry of works optimized [42, 102] and generalized [16, 46] the original construction to be suitable for modern applications. These proofs are heavily influential for two reasons: First, they served as a fundamental stepping stone for constructing general-purpose zkSNARKs without a trusted setup [54, 124, 136]. Second, they were the first to hint at a new style of interaction in which the verifier does not immediately resolve the prover's statement to true or false, but rather reduces it to a simpler statement to be checked. As we will discuss, this has become the predominant paradigm for interactions today.

Efficient proofs for inner-products enabled a majority of the general-purpose zkSNARKs with less trust in the setup process. Initial proof systems in this vein such as Sonic [108], Plonk [73], and Marlin [54] explored a universal setup (i.e., a trusted setup is only needed once for all computations). The question of an entirely trustless setup, however, was answered by the Spartan proof system [124]: Spartan was the first to show that it is possible to achieve an untrusted setup but still achieve a succinct verifier for general-purpose computations, bringing us to the state-of-the-art for proof systems at the turn of the decade.

Reminiscent the original inner-product proof, of these works are characterized by a new design paradigm, which fit together a handful of well-studied *reductions*, each of which reduced the original statement into a simpler statement to be checked. As a particularly salient example, Ràfols and Zapico [120] observe that nearly all modern SNARKs rely on a "compression" step which reduces the task of checking many algebraic constraints (representing a computation) to the task of checking a single constraint using a random

linear combination. This single constraint can subsequently be checked using, say, an inner-product proof.

1.1.6 Modern Proofs of Knowledge

At the beginning of the 2020s, we had general-purpose proof systems [54, 73, 108, 124] that were capable of efficiently proving various (hand optimized) programs such as blockchain translation validation [122], data structure queries [132, 142], and anonymous credentials [63]. However, this required significant development effort, as every new application warranted the development of a newly optimized proof system. In response, modern interest has centered around a particularly compelling application: proving the correct execution of a *virtual machine* [6, 7, 8, 9]. This enables a single-proof system for the underlying substrate running all applications.

The problem of proving virtual machine execution reduces to the more fundamental problem of proving recursive applications of a function F. In particular, we are concerned with proving statements of the form $F^n(z_0) = z_n$, for some input z_0 and some output z_n, where F^n denotes applying F for n times. The function F can also take secret auxiliary inputs in each step, which are hidden in the final proof. In the case of virtual machines, F can represent a single step of execution (e.g., a single cycle of the CPU).

Historically, the best-known approach to design a proof system for recursive applications of a function F was to unroll the entire execution $F \circ F \circ \cdots \circ F$ into a monolithic arithmetic circuit, and then use a standard proof system with short proofs for circuit satisfiability [25]. Unfortunately, the prover's memory overhead would scale with the *entire trace* of the computation. Moreover, this bounds the recursion depth ahead of time.

The first breakthrough surprisingly occurred a decade prior in 2012, when Valiant [133] proposed incrementally verifiable computation (IVC), which reflected the recursive structure of the computation into the proof itself: Given a short proof π_i attesting to i steps of computation, the prover can write a short proof π_{i+1} that attests to $i+1$ steps by proving the correct execution of an arithmetic circuit that runs the latest step of computation, and checks π_i (using the proof system's verifier). This avoids having to fix the recursion depth ahead of time, while ensuring that the prover's memory overhead only scales with a single step of execution. Unfortunately, Valiant's technique requires having to represent a proof system's verifier in an arithmetic circuit, which is prohibitively expensive in practice due to having to represent the underlying elliptic curve operations.

Since Valiant, research effort has largely focused on reducing the recursion overhead (i.e., the cost of verifying proofs in an arithmetic circuit). The pursuit of zero-knowledge proof systems that are amenable to efficiently proving statements about their own validity has fostered a remarkably fertile research program in the past decade [25, 27, 30, 38, 45, 55, 101, 133].

In 2017, Ben-Sasson et al. [27] provide the first major improvement to Valiant's original construction. In particular, they demonstrate that they can use a *cycle of elliptic curves*, where operations on one curve can be represented efficiently as constraints on the other curve. In this way, we can efficiently represent a a verifier that performs operations on the first curve as a circuit over the second curve and visa-versa. While cycles of curves

brought Valiant's approach to the realm of feasibility, we were still far from the realm of practicality.

The big leap in recursive proof design was due to Bowe, Grigg, and Hopwood [38] in 2019, who observed that it is sufficient for the verifier circuit to partially check the previous proof and then defer the rest into a running proof (which does not grow in size) that can be checked at the end. Bowe et al. observe that *reducing* the task of checking a running proof and a fresh proof into the task of checking an updated running proof is significantly cheaper than actually checking the proof, bringing IVC into the realm of practicality. Once again, a reduction, rather than a direct proof proved to be the key ingredient for advancing the theory. This has lead to a modern influx of optimized reductions for this setting [33, 41, 65, 97, 99, 101, 146], cementing them as a core building block for modern proofs.

Taking a sweeping look, zkSNARKs today are developing much like how computer architecture developed half a century ago, transitioning from domain specific applications to highly-optimized general purpose machines. Central to this development is a new paradigm shift from developing interactive proofs to developing interactive *reductions*. This observation provides the starting point for this thesis.

1.2 Summary of Contributions

As discussed in Section 1.1, proofs of knowledge [83] are powerful cryptographic primitives that allow a verifier to efficiently check (in zero-knowledge) that a prover *knows* a satisfying witness for a claimed statement. Such proofs provide strong integrity and privacy guarantees that enable a large class of cryptographic applications [63, 95, 122, 140].

However, a growing body of work challenges the traditional paradigm by describing interactions in which the verifier does not fully resolve the prover's statement to true or false, but rather reduces it to a simpler statement to be checked:

- The well-studied *inner-product argument* [36] (along with subsequent optimizations [42] and generalizations [37, 46]) relies on recursively applying an interactive reduction from the task of checking knowledge of size n vectors to the task of checking knowledge of size $n/2$ vectors.

- *Aggregation schemes* for polynomial commitments [35, 38] and *unbounded aggregation schemes* for linear-map vector commitments [49] can both be viewed as interactive reductions from checking proofs of several openings to a commitment to checking a proof of a single opening to a commitment.

- *Split-accumulation schemes* [45] can be viewed as interactive reductions from checking several proofs of knowledge and several accumulators to checking a single accumulator.

- Incrementally verifiable computation reduces the task of checking a succinct proof of n applications of function F and a succinct proof of m subsequent applications of F to the task of checking a succinct proof of $n + m$ applications of F [133].

- As observed by Ràfols and Zapico [120], most proof systems with a universal and updatable trusted setup (e.g., [48, 54, 100, 124]) construct an interactive reduction from the task of checking knowledge of a preimage of a matrix evaluation to the task of checking knowledge of a preimage of a vector evaluation.

Such interactive reductions, although central to modern proofs of knowledge, lack a unifying theoretical foundation. As evidenced above, these reductions typically have case-by-case security definitions (if any at all) that are tailored towards the larger systems that rely on them. The lack of a common language makes it difficult to relate comparable techniques hidden under incomparable abstractions. Moreover, stitching together various techniques requires remarkably delicate (and often tedious) reasoning for how the soundness of the larger protocol reduces to the soundness of each subprotocol.

Towards a unifying language, we formalize the notion of an interactive reduction over statements of knowledge, in which the verifier reduces the task of checking the original statement to the task of checking a new (simpler) statement. We refer to such a protocol as a *reduction of knowledge*. Reductions of knowledge formalize our central thesis statement that proofs of knowledge are maps between propositions of knowledge.

The goal of this thesis is to formally develop this insight and its consequences. A key observation is that reductions of knowledge serve as more than just a crisp abstraction: We prove that reductions of knowledge can be composed sequentially or in parallel, and thus can be stitched together to construct complex proofs of knowledge. Therefore, reductions of knowledge can be viewed as a theory of composition for proofs of knowledge. In Chapter 2, we formally develop reductions of knowledge and the corresponding composition results.

Given an ambient framework, we present the following new constructions in our framework. A recurring theme is that reductions of knowledge are particularly effective at reasoning about recursion.

- In Chapter 3, we present *tensor reductions of knowledge* as a generalization of the core reductive step in most recursive algebraic proofs. By instantiating and recursively composing the tensor reduction of knowledge over appropriate spaces, we derive both new and existing proofs of knowledge for various linear algebraic structures [16, 35, 36, 37]. Tensor reductions of knowledge afford a unified theory for proofs of knowledge in this class.

- In Chapter 4, we present *folding schemes*, which are efficient reductions of knowledge from the task of checking two instances in a relation to checking a single instance in a relation. Folding schemes provide a minimal abstraction for various protocols in the literature [36, 114, 121]. We present several folding schemes for NP instances.

- In Chapter 5, we construct a general compiler from a folding scheme for NP to an *incrementally verifiable computation* (IVC [133]) scheme. The resulting IVC schemes remain the fastest to date with various tradeoffs. For a non-deterministic function F, IVC can be understood as a non-interactive reduction of knowledge from the task of checking a succinct proof of n applications of F and a succinct proof of m subsequent applications of F to the task of checking a succinct proof of $n + m$ applications of

F. To formally capture this intuition, we introduce *refined reductions of knowledge*, which additionally constrain the input and output statements of a reduction. We additionally extend our approach for IVC to derive a scheme for *non-uniform* IVC which is designed to efficiently prove virtual machine computations.

- In Chapter 6, we present several proofs of knowledge for NP as a sequence of reductions, each with unique overheads and assumptions.

Just as standard reductions are used for principled algorithm design, reductions of knowledge are intended for principled proof design. Throughout our development, we provide various instructional examples to demonstrate how reductions of knowledge offer a promising route towards taming the complexity of modern proofs.

While reductions of knowledge help build concrete proof systems, they also gives us a coherent language to develop the theory of proofs of knowledge in general. Given this language, We develop broad results which apply to large classes of constructions formalized in our framework:

- In Chapter 7, we develop several transformations that imbue broad classes of reductions of knowledge with stronger properties. First, a central challenge with reductions of knowledge, is that they can only be composed a constant number of times without additional knowledge assumptions, due to an exponential blowup in the extractor runtime [30]. To achieve only a polynomial runtime blowup, we adapt Fischlin's transformation [69] for reductions of knowledge to achieve straight-line extractors. Second, As zero-knowledge (i.e., proofs do not reveal any information about the witness) is a cornerstone property in both theory and practice, we additionally develop a notion of *zero-knowledge* reductions of knowledge and proving closure under sequential and parallel composition. We additionally design a generic transformation from any reduction of knowledge to a zero-knowledge reduction of knowledge.

- In Chapter 8, we take a sweeping look at everything we have developed in this thesis. We use the language of category theory to organize various concepts and derive broad insights into the structure of proofs of knowledge as a whole.

The remainder of this chapter provides a technical overview of this thesis. In Section 1.3, we provide an informal overview of reductions of knowledge and the corresponding composition results. In Section 1.4, we overview tensor reductions of knowledge, and provide a vector commitment proof of knowledge as a concrete example. In Section 1.5, we overview a general compiler from a folding scheme for NP to an IVC scheme.

1.3 Reductions of Knowledge

Recall that proofs of knowledge are defined over a relation $\mathcal{R}$ and allow a prover to show for some statement u that it knows witness w such that $(u, w) \in \mathcal{R}$. In contrast, a reduction of knowledge is defined over a pair of relations $\mathcal{R}_1$ and $\mathcal{R}_2$, and enables a verifier to reduce

the task of checking knowledge of a satisfying witness for a statement in $\mathcal{R}_1$ to the task of checking knowledge of a satisfying witness for a new statement in $\mathcal{R}_2$.

Definition 1.1 (Reduction of Knowledge, Informal). A reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ is an interactive protocol between a prover and a verifier. Both parties take as input a claimed statement u_1 to be checked, and the prover additionally takes as input a corresponding witness w_1 such that $(u_1, w_1) \in \mathcal{R}_1$. After interaction, the prover and verifier together output a new statement u_2 to be checked in place of the original statement, and the prover additionally outputs a corresponding witness w_2 such that $(u_2, w_2) \in \mathcal{R}_2$. A reduction of knowledge satisfies the following properties.

(i) Completeness: If the prover is provided a satisfying witness w_1 for the verifier's input statement u_1, then the prover outputs a satisfying witness w_2 for the verifier's output statement u_2.

(ii) Knowledge Soundness: If an arbitrary prover provides a satisfying witness w_2 for the verifier's output statement u_2, then the prover almost certainly knows a satisfying witness w_1 for the verifier's input statement u_1.

We write $\Pi : \mathcal{R}_1 \to \mathcal{R}_2$ to denote that protocol Π is a reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$. Alternatively, we say that Π has type $\mathcal{R}_1 \to \mathcal{R}_2$.

There are two ways to conceptually reconcile reductions of knowledge with proofs of knowledge. First, proofs of knowledge can be viewed as a special case of reductions of knowledge where the second relation $\mathcal{R}_2$ is fixed to encode true or false. This interpretation helps naturally translate existing tooling used to study proofs of knowledge to study reductions of knowledge. For instance, we can expect reductions of knowledge to be compatible with idealized soundness models such as the random oracle model [21] and the algebraic group model [71], idealized communication models such as interactive oracle proofs [26] and variants [44, 48, 54], and heuristic transformations such as Fiat-Shamir [67].

Second, reductions of knowledge can be interpreted as proofs for *conditional* statements in which a prover shows for some u_1 that it knows w_1 such that $(u_1, w_1) \in \mathcal{R}_1$ *contingent* on the fact that for u_2 output by the verifier it knows w_2 such that $(u_2, w_2) \in \mathcal{R}_2$. Put more plainly, reductions of knowledge are proofs for statements of the form "If you believe that I know a witness for statement u_2 in $\mathcal{R}_2$, then you should believe that I know a witness for statement u_1 in $\mathcal{R}_1$". This interpretation helps characterize statements that reductions of knowledge can handle more naturally than proofs of knowledge.

Reductions of knowledge can also be viewed as a probabilistic variant of Levin reductions [12] (i.e., Karp reductions [12] that map witnesses as well as statements) that verifiably proceed through interaction. Under this interpretation, Levin reductions can be understood as deterministic reductions of knowledge with no interaction.

Under any interpretation, we are interested in proving that reductions of knowledge can be composed sequentially and in parallel. Such a requirement holds immediately for standard notions of reductions, but requires subtle reasoning when considering knowledge soundness: To ensure that sequential composability holds, we additionally require that reductions of knowledge are *publicly reducible*. That is, given the input statement u_1 and

the interaction transcript, any party should be able to reconstruct the output statement u_2. As we detail in Section 2.1, this seemingly innocuous requirement becomes the linchpin in arguing sequential composability. With public reducibility, we have the following.

Theorem 1.1 (Sequential Composition, Informal). Consider relations $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$. For reductions of knowledge $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3$ we have that $\Pi_2 \circ \Pi_1$ is a reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_3$ where $\Pi_2 \circ \Pi_1$ denotes the protocol that first runs Π_1, and then runs Π_2 on the statement and witness output by Π_1.

By parallel composition, we do not mean running both protocols at the same time, but rather that the composed protocol takes as input instance-witness pairs in parallel and and outputs instance-witness pairs in parallel. For relations $\mathcal{R}_1$ and $\mathcal{R}_2$, let relation $\mathcal{R}_1 \times \mathcal{R}_2$ be such that $((u_1, u_2), (w_1, w_2)) \in \mathcal{R}_1 \times \mathcal{R}_2$ if and only if $(u_1, w_1) \in \mathcal{R}_1$ and $(u_2, w_2) \in \mathcal{R}_2$. Then, we have the following.

Theorem 1.2 (Parallel Composition, Informal). Consider relations $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_4$. For reductions of knowledge $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 : \mathcal{R}_3 \to \mathcal{R}_4$ we have that $\Pi_1 \times \Pi_2$ is a reduction of knowledge from $\mathcal{R}_1 \times \mathcal{R}_3$ to $\mathcal{R}_2 \times \mathcal{R}_4$ where $\Pi_1 \times \Pi_2$ denotes the protocol that runs Π_1 on the statement-witness pair in $\mathcal{R}_1$, runs Π_2 on the statement-witness pair in $\mathcal{R}_3$, and outputs the pair of results.

We now proceed to develop reductions of knowledge in more detail. In Section 1.3.1, we explain how reductions of knowledge form a compositional framework for proofs of knowledge. In Section 1.3.2, we provide a concrete example of how our composition results can be used to construct complex reductions of knowledge. Finally, in Section 1.3.3, We then observe that a more restricted — but simpler — notion of soundness, known as tree extractability, implies our definition of knowledge soundness (Lemma 2.2).

1.3.1 A Compositional Framework for Proofs of Knowledge

Reductions of knowledge can be viewed as a minimal compositional framework that can feasibly capture and tame the growing complexity of modern proofs of knowledge. Regardless of how reductions are stitched together, our composition results abstract out the pedantic reasoning for how exactly to use the soundness of each subcomponent to prove the soundness of the composed reduction.

In more detail, the requirement that the prover *knows* a witness is formally stated as an extractability property: Given an expected polynomial-time prover that can produce a satisfying interaction, there must exist a corresponding expected polynomial-time extractor that can extract the alleged witness (e.g., by running and rewinding the prover internally). This definition, while undoubtedly natural, requires subtle reasoning when constructing large proofs of knowledge which rely on several sub-proofs: In general, the soundness analysis must meticulously detail how to use the successful prover to construct successful provers for each sub-proof and then use the corresponding extractors to derive an extractor for the overall proof.

In the public-coin setting (where the verifier only sends random challenges), Bootle et al. [36] abstract away some low-level reasoning by proving that *tree special soundness*

implies the standard notion of knowledge soundness. Tree special soundness holds when a *tree of accepting transcripts* contains sufficient information to reconstruct the witness, with each path representing a unique transcript and each branch representing diverging verifier randomness. Both Lee [102] and Attema and Cramer [16] show that tree special soundness implies modularity by observing that tree special sound protocols can be sequentially composed to produce a tree special sound protocol.

As demonstrated by these works, tree special soundness is a remarkably useful abstraction for simplifying sequentially composed, uniformly structured proofs of knowledge (e.g., proofs that recursively invoke themselves). However, when dealing with larger proofs of knowledge that invoke various *independent* sub-proofs, such as modern proofs for NP, tree special soundness is no longer an appropriate abstraction: having a single transcript that weaves through all such sub-proofs and globally forks with each local challenge undermines the intended semantics and unnecessarily blows up the knowledge error (i.e., the extractor's failure probability).

Reductions of knowledge are designed precisely to reason about such proofs. Unlike prior work, our parallel composition operator enables us to capture proofs of knowledge with arbitrary dependence topologies. For instance, most proof systems for NP, such as Spartan [124], Poppins [100], and Marlin [54], reduce a statement in an NP-complete relation such as R1CS [78] to several simpler linear algebraic statements (such as inner-product and polynomial evaluation claims), each of which is then checked using a tailored proof of knowledge [120]. As a concrete example, we show that an proof of knowledge for NP can be captured modularly in our framework by utilizing both sequential and parallel composition.

Moreover, because we demonstrate that *any* two publicly verifiable reductions can be composed, this opens up the ability to modularly reason about knowledge-assumption-based succinct non-interactive arguments of knowledge (SNARKs [29, 79]) and incrementally verifiable computation [133], which currently fall back on composing extractors in intricate ways [45, 100, 101]. As a concrete example, we demonstrate how to succinctly express non-interactive ℓ-*folding schemes* [101, 121] (i.e, folding schemes reducing ℓ initial instances) by utilizing a tree-like dependence topology in our reductions of knowledge framework.

In the public-coin setting, we incorporate prior progress into our framework by proving that tree special soundness implies our notion of knowledge soundness. As such, public-coin reductions can be analyzed using standard techniques.

1.3.2 Example: Folding Schemes

We now provide a concrete example to demonstrate the utility of our composition results. In particular, we study folding schemes, which are interactive protocols that reduces the task of checking two instances in a relation to the task of checking a single instance in the relation. Folding schemes provide a minimal abstraction for various protocols in the literature.

Recently, Ràfols and Zacharakis [121] provide non-interactive ℓ-folding schemes (i.e., folding schemes for ℓ initial statements) for the vector commitment relation, inner-product

relation, and polynomial commitment relation. Such folding schemes help amortize the verifier's work over multiple instances in larger non-interactive proofs of knowledge, which typically involve checking multiple instances of the same form.

As these folding schemes rely on knowledge assumptions rather than interaction, prior techniques cannot help modularize the corresponding soundness analysis. As promised, we can still achieve modularity by decomposing them as a sequence of *non-interactive* reductions of knowledge. Formally, a non-interactive reduction of knowledge is one in which the interaction only consists of messages from the prover. Non-interactive ℓ-folding schemes can be succinctly formalized as a particular class of non-interactive reductions of knowledge. Letting $\mathcal{R}^\ell$ denote $\mathcal{R} \times \ldots \times \mathcal{R}$ for ℓ times, we define the following.

Definition 1.2 (ℓ-Folding Schemes). A (non-interactive) ℓ-folding scheme for relation $\mathcal{R}$ is a (non-interactive) reduction of knowledge from $\mathcal{R}^\ell$ to $\mathcal{R}$.

Ràfols and Zacharakis achieve ℓ-folding schemes for various relations by recursively composing 2-folding schemes in a tree-like fashion. In particular, ℓ instances are treated as leaves in a tree. A 2-folding scheme is then used to fold each pair of adjacent instances to produce a total of $\ell/2$ instances. These $\ell/2$ instances are once more folded in a pairwise fashion to produce $\ell/4$ instances and so on until a single instance remains.

As demonstrated by Ràfols and Zacharakis, while the tree-folding protocol can be stated in a straightforward manner, the corresponding knowledge soundness analysis requires careful attention to detail. In particular, the corresponding proof involves demonstrating that the malicious prover induces a corresponding expected polynomial-time extractor that unfolds once. Such an extractor is then shown to induce a pair of expected polynomial-time malicious provers for the previous layer of the tree, and so on. Alternatively, by working in the reductions of knowledge framework, nearly all of this reasoning is abstracted away. Indeed, we condense the original three-page proof into several lines.

Lemma 1.1 (ℓ-Folding Scheme). Consider a (non-interactive) 2-folding scheme Π_{TF} for relation $\mathcal{R}$ and $\ell = 2^i$ for $i \in \mathbb{N}$ where $i \geq 1$. Then, Π_ℓ, inductively defined as follows, is a (non-interactive) ℓ-folding scheme for $\mathcal{R}$.

$$\Pi_\ell = \Pi_{\mathsf{TF}} \circ (\Pi_{\ell/2} \times \Pi_{\ell/2})$$
$$\Pi_2 = \Pi_{\mathsf{TF}}$$

Proof. We reason inductively over i. In the base case, suppose $i = 1$. Then, by construction, Π_2 is a 2-folding scheme. Suppose instead $i \geq 2$. Suppose that for $\ell = 2^i$ we have that $\Pi_{\ell/2}$ is a $(\ell/2)$-folding scheme. Then, $\Pi_{\ell/2} \times \Pi_{\ell/2}$ is a reduction of knowledge from $\mathcal{R}^{\ell/2} \times \mathcal{R}^{\ell/2} = \mathcal{R}^\ell$ to $\mathcal{R}^2$. Thus, $\Pi_{\mathsf{TF}} \circ (\Pi_{\ell/2} \times \Pi_{\ell/2})$ is a reduction of knowledge from $\mathcal{R}^\ell$ to $\mathcal{R}$. □

1.3.3 Knowledge Soundness from Tree Extraction

When proving constructions secure, reasoning about knowledge soundness directly is typically cumbersome. To alleviate this issue, prior work [36] observes that most protocols are

algebraic: The corresponding extractor typically runs the malicious prover multiple times with refreshed verifier randomness to retrieve accepting transcripts, which can be interpolated to retrieve the witness. Leveraging this insight, Bootle et al. [36] provide a general extraction lemma, which states that to prove knowledge soundness for algebraic protocols, it is sufficient to show that there exists an extractor that can produce a satisfying witness when provided a *tree of accepting transcripts* with refreshed verifier randomness at each layer. This proof technique has been adapted to various settings [37, 42, 45, 101], and we similarly provide the corresponding lemma for reductions of knowledge.

Definition 1.3 (Tree of Transcripts). Consider an m-round public-coin interactive protocol $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ that satisfies the interface described in Definition 2.1. A $(n_1, \ldots, n_m)$-tree of accepting transcripts for statement u_1 is a tree of depth m where each vertex at layer i has n_i outgoing edges such that (1) each vertex in layer $i \in [m]$ is labeled with a prover message for round i; (2) each outgoing edge from layer $i \in [m]$ is labeled with a different choice of verifier randomness for round i; (3) each leaf is labeled with an accepting statement-witness pair output by the prover and verifier corresponding to the interaction along the path.

Lemma 1.2 (Tree Extraction [37]). Consider an m-round public-coin interactive protocol $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ that satisfies the interface described in Definition 2.1 and satisfies completeness. Then $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a reduction of knowledge if there exists a PPT extractor χ that, for all instances u_1, outputs a satisfying witness w_1 with probability $1 - \mathsf{negl}(\lambda)$, given an $(n_1, \ldots, n_m)$-tree of accepting transcripts for u_1 where the verifier's randomness is sampled from space Q such that $|Q| = O(2^\lambda)$, and $\prod_i n_i = \mathsf{poly}(\lambda)$.

Proof Intuition. Our proof closely follows that of Bootle et al. [36]. At a high level, we construct an expected polynomial-time extractor $\mathcal{E}$ that repeatedly runs the malicious prover $\mathcal{P}^*$ and collects corresponding accepting transcripts and associated output statement-witness pairs. The extractor then passes these collected transcripts to χ which retrieves the desired witness by assumption. □

1.4 Recursive Algebraic Proofs of Knowledge

Reductions of knowledge provide the necessary abstraction to view various techniques under a unifying lens. As a demonstration, we consolidate recursive proofs over homomorphic structures by recasting their central recursive step as instantiations of the tensor reduction of knowledge, which we introduce below.

In more detail, modern proofs of knowledge are designed around leveraging homomorphic structure to achieve better asymptotics and concrete efficiency. An influential line of work [16, 17, 36, 42, 102] studies the consequences of proofs over structurally nested homomorphic objects such as vectors, matrices and hypercubes. A key insight is that such objects contain sufficient algebraic structure for *recursive* proofs in which larger composed statements can be reduced to smaller constituent statements of the same form. For instance, Bootle et al. [36] show that the task of checking an inner-product over committed

size n vectors can be split into the task of checking two inner-products over committed size $n/2$ vectors which can then be "folded" into the task of checking a single inner-product over committed size $n/2$ vectors. Homomorphic structures that enable recursive techniques have become a staple in constructing efficient proof systems for NP [42, 100, 124, 136]. However, while proofs over recursive homomorphic structures have become an essential tool in practice, the literature detailing such techniques is becoming increasingly dissonant with sparse progress on unifying the disparate approaches.

Bünz et al. [46] initiate the study of a unified theory by observing that existing inner-product proofs [36, 42] only require a commitment scheme that is homomorphic over both the commitment keys and messages. Thus, such inner-product proofs can be viewed as instantiations of a generic inner-product proof that only leverages these properties. Bootle, et al. [37] further relax this requirement by observing that split-and-fold style techniques in general [16, 42, 44, 46] only require a commitment scheme that can be computed by summing over a hypercube. Leveraging this insight, Bootle et al. show that such techniques can be interpreted as instantiations of the familiar sum-check protocol [106].

We considerably sharpen the sufficient conditions with the following observation: Protocols such as the sumcheck protocol and the inner-product proof only require the underlying linear-algebraic objects (e.g., polynomials, vectors, and matrices) to form a module (i.e., have a notion of addition and scalar multiplication). Abstracting away the specific details of the associated modules, all such protocols reduce a claim in a "tensored" module to claims in constituent modules. Leveraging this insight, we design an information-theoretic protocol, the *tensor reduction*, as a sweeping generalization of protocols in this class. Conceptually, the tensor reduction explains why such a broad class of protocols look different but feel the same.

Theorem 1.3 (Tensor Reduction, Informal). For modules U, U_1, and U_2 such that $U \cong U_1 \otimes U_2$, there exists an interactive reduction that reduces the task of evaluating a homomorphism in U to the task of evaluating a homomorphism in U_1 and evaluating a homomorphism in U_2.

Essentially, the versatility of the tensor reduction stems from its ability to work over any pair of modules and any valid notion of a tensor product between these modules. In particular, the tensor product can be defined as *any* operator that satisfies the prescribed universality property: the tensor product of any two modules U_1 and U_2 must result in a new module, denoted $U_1 \otimes U_2$, such that any bilinear mapping $\varphi : U_1 \times U_2 \to V$ induces a unique homomorphism $\widetilde{\varphi} : U_1 \otimes U_2 \to V$ such that $\widetilde{\varphi}(u_1 \otimes u_2) = \varphi(u_1, u_2)$.

For instance, for field $\mathbb{F}$, let the tensor product denote the outer product and consider an arbitrary vector in $\mathbb{F}^n$. This vector can be interpreted as a matrix in $\mathbb{F}^{(n/2)\times 2}$ or equivalently as an element of $\mathbb{F}^{n/2} \otimes \mathbb{F}^2$ which consists of sums of outer products of vectors in $\mathbb{F}^{n/2}$ and $\mathbb{F}^2$. Thus, the tensor reduction can reduce a claim over a vector in $\mathbb{F}^n$ to a claim over a vector in $\mathbb{F}^{n/2}$ and a vector in $\mathbb{F}^2$. Similarly, by taking the tensor product to be polynomial multiplication, the tensor reduction can reduce a claim over a degree (m, n) bivariate polynomial in $\mathbb{F}[X, Y] \cong \mathbb{F}[X] \otimes \mathbb{F}[Y]$ to a claim over a degree m univariate polynomial in $\mathbb{F}[X]$ and a degree n univariate polynomial in $\mathbb{F}[Y]$. By taking the tensor product to be the Kronecker product, the tensor reduction can reduce a claim over a matrix in $\mathbb{F}^{mp\times nq}$

to a claim over a matrix in $\mathbb{F}^{m\times n}$ and a matrix in $\mathbb{F}^{p\times q}$. By taking the tensor product to be a pairing operation mapping groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to $\mathbb{G}_\mathsf{T}$, the tensor reduction can reduce a claim over $\mathbb{G}_\mathsf{T}$ to claims over $\mathbb{G}_1$ and $\mathbb{G}_2$.

Just as the sum-check protocol can be used to design proofs of knowledge, the tensor reduction can be used to design reductions of knowledge. By instantiating the tensor reduction over vector spaces, we derive the *tensor reduction of knowledge*, an unconditionally secure protocol that generalizes the central reductive step common to most recursive algebraic proofs.

Theorem 1.4 (Tensor Reduction of Knowledge, Informal). For $\hom(W, V)$, denoting homomorphisms from vector space W to vector space V, and length n, there exists a reduction of knowledge that reduces the task of checking knowledge of $w \in W^n$ such that $u(w) = v$ for $u \in \hom(W^n, V)$ and $v \in V$ to the task of checking knowledge of $w' \in W$ such that $u'(w') = v'$ for $u' \in \hom(W, V)$ and $v' \in V$.

In Chapter 3, we leverage the above composition result, to show that tensor reductions of knowledge can be recursively composed to recover various recursive proofs. In particular, we appropriately instantiate the vector spaces to recover a family of reductions of knowledge for vector commitments [35, 36, 37] and linear forms [16, 17].

We additionally develop a new family of proofs for bilinear forms which falls out naturally from our prior generalizations. In particular, consider prime order group $\mathbb{G}$ and corresponding scalar field $\mathbb{F}$. For public key $G \in \mathbb{G}^m$, public matrix $M \in \mathbb{F}^{m\times m}$, commitments $\overline{A}, \overline{B} \in \mathbb{G}$, and scalar $\sigma \in \mathbb{F}$, a bilinear forms proof allows a verifier to check that a prover knows $A, B \in \mathbb{F}^m$ such that $A^\top M B = \sigma$, $\langle G, A\rangle = \overline{A}$ (i.e., the inner-product of G and A is $\overline{A}$), and $\langle G, B\rangle = \overline{B}$.

In practice, the matrix M in the bilinear forms relation can encode a variety of constraints. For instance, if M is the identity matrix then the verifier can check the inner-product of A and B (and more generally the inner product of any rearrangement of A and B). If instead M assigns weights to the diagonal, then the verifier can check a weighted inner-product [46, 58]. More generally, M can encode any degree-two custom-gate [73], enabling an expressive constraint system for NP.

1.4.1 Example: A Vector Commitment Proof

Rather than presenting tensor reduction of knowledge in the fully abstracted setting, we present the *vector commitment proof* [36], which results from instantiating tensor reduction of knowledge over vector commitments. The vector commitment proof concretely intuits the underlying mechanics of the tensor reduction of knowledge.

The vector commitment proof allows a prover to show that it knows the opening to a Pedersen vector commitment [119]. In more detail, consider group $\mathbb{G}$ of prime order p and corresponding scalar field $\mathbb{F} = \mathbb{Z}_p$. Consider some public key $G \in \mathbb{G}^n$ where $n = 2^i$ for some $i \in \mathbb{N}$. Suppose a prover would like to *succinctly* demonstrate to a verifier that it knows $A \in \mathbb{F}^n$ such that $\langle G, A\rangle = \overline{A}$ (i.e., the inner-product of G and A is $\overline{A}$). That is, we would like to design a proof of knowledge for the following relation.

Definition 1.4 (Vector Commitment Relation). The vector commitment relation is defined as $\mathcal{R}_{\mathsf{VC}}(n) = \{((G, \overline{A}), A) \in ((\mathbb{G}^n, \mathbb{G}), \mathbb{F}^n) \mid \langle G, A\rangle = \overline{A}\}$.

The vector commitment relation, can be immediately treated as a claim that is reducible by the tensor reduction of knowledge. In particular, we can treat the commitment keys as a homomorphism in $\mathbb{G}^n \cong \hom(\mathbb{F}^n, \mathbb{G})$, mapping the underlying message in $\mathbb{F}^n$ to the commitment space $\mathbb{G}$. Then, the tensor reduction of knowledge can reduce a claim about a preimage of a homomorphism in $\mathbb{G}^n \cong (\mathbb{G}^{n/2})^2$ to a claim about a preimage of a homomorphism in $\mathbb{G}^{n/2}$. We can recursively compose such a reduction to achieve a proof of knowledge for the vector commitment relation. We now describe the resulting proof of knowledge concretely.

At a high level, the verifier splits the task of checking knowledge of vector A into the task of checking knowledge of the first and second half of A. Instead of checking each separately, the verifier "folds" the two checks into a single check using a random linear combination. The prover computes the corresponding random linear combination of the first and second half of A to produce a folded witness vector that is half the original size. This folding procedure is recursively run until the length of the vector to be checked is 1. At this point the prover directly sends the vector to the verifier.

We start by designing a reduction of knowledge that reduces the task of checking knowledge of a size n vector to checking knowledge of a size $n/2$ vector.

Construction 1.1 (Vector Commitment Reduction of Knowledge). We construct a reduction of knowledge from $\mathcal{R}_{\mathsf{VC}}(n)$ to $\mathcal{R}_{\mathsf{VC}}(n/2)$ for $n = 2^i$ where $i \geq 1$. Suppose that the prover $\mathcal{P}$ and verifier $\mathcal{V}$ take as input statement $(G, \overline{A}) \in (\mathbb{G}^n, \mathbb{G})$ and that the prover additionally takes as input alleged witness vector $A \in \mathbb{F}^n$ such that

$$((G, \overline{A}), A) \in \mathcal{R}_{\mathsf{VC}}(n).$$

The reduction proceeds as follows.

1. $\mathcal{P}$: Let G_1 and G_2 (respectively A_1 and A_2) denote the first and second half of vector G (respectively A). The prover begins by sending $\overline{A}_{ij} \leftarrow \langle G_i, A_j\rangle$ for $i, j \in \{1, 2\}$. Here, $\overline{A}_{11}$ and $\overline{A}_{22}$ represent the first and second "half" of the original commitment $\overline{A}$, and $\overline{A}_{12}$ and $\overline{A}_{21}$ represent cross terms which will assist the verifier in folding the original statement.

2. $\mathcal{V}$: The verifier first checks the consistency of $\overline{A}_{11}$ and $\overline{A}_{22}$ with $\overline{A}$ by checking that $\overline{A}_{11} + \overline{A}_{22} = \overline{A}$. The verifier must still check that the prover knows A_1 and A_2 such that $\overline{A}_{11} = \langle G_1, A_1\rangle$ and $\overline{A}_{22} = \langle G_2, A_2\rangle$. Instead of checking each individually, the verifier folds them into a single check by using a random linear combination. In particular, the verifier sends random $r \in \mathbb{F}$ to $\mathcal{P}$.

3. $\mathcal{P}, \mathcal{V}$: Together, the prover and verifier output the folded key and corresponding commitment $(G', \overline{A}') \in (\mathbb{G}^{n/2}, \mathbb{G})$ where $G' \leftarrow G_1 + r \cdot G_2$ and $\overline{A}' \leftarrow \overline{A}_{11} + r \cdot (\overline{A}_{12} + \overline{A}_{21}) + r^2 \cdot \overline{A}_{22}$.

4. $\mathcal{P}$: The prover outputs the folded witness $A' \in \mathbb{F}^{n/2}$ where $A' \leftarrow A_1 + r \cdot A_2$.

Now, to check the original statement, it is sufficient for the verifier to check that the prover knows A' such that

$$((G', \overline{A}'), A') \in \mathcal{R}_{\mathsf{VC}}(n/2).$$

To prove knowledge soundness, we must show that given a prover that produces a witness for the output statement with non-negligible probability, we can derive an extractor that can use this prover to derive a witness for the input statement with nearly the same probability. Because the above reduction is public-coin, it suffices to show that there exists an extractor that can derive a satisfying input witness given a tree of transcripts and corresponding satisfying outputs (Lemma 2.2). Intuitively, the original extractor can generate such a tree by repeatedly rewinding the prover and collecting transcripts in which the prover outputs a satisfying witness.

Lemma 1.3 (Vector Commitment Reduction of Knowledge). For $n = 2^i$ where $i \geq 1$, Construction 1.1 is a reduction from $\mathcal{R}_{\mathsf{VC}}(n)$ to $\mathcal{R}_{\mathsf{VC}}(n/2)$.

Proof. We reason via tree extractability (Lemma 2.2). Suppose an extractor is provided with a tree of transcripts which consists of three transcripts, where the kth transcript has the same initial message $\overline{A}_{ij}$ for $i, j \in \{1, 2\}$, random challenge r_k, and satisfying output instance-witness pairs

$$((G'_k, \overline{A}'_k), A'_k) \in \mathcal{R}_{\mathsf{VC}}(n/2). \tag{1.2}$$

The extractor first solves for a_k for $k \in \{1, 2, 3\}$ such that

$$\begin{pmatrix} 1 & 1 & 1 \\ r_1 & r_2 & r_3 \\ r_1^2 & r_2^2 & r_3^3 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \tag{1.3}$$

using an inverse Vandermonde matrix. The extractor then computes and outputs the unfolded witness

$$A = \Big(\sum_k a_k \cdot A'_k, \sum_k a_k r_k \cdot A'_k \Big). \tag{1.4}$$

Indeed, we have that

$$\begin{aligned}
\langle G, A \rangle &= \Big\langle G_1, \sum_k a_k \cdot A'_k \Big\rangle + \Big\langle G_2, \sum_k a_k r_k \cdot A'_k \Big\rangle && \text{By Equation (1.4).} \\
&= \sum_k a_k \cdot \langle G_1 + r_k \cdot G_2, A'_k \rangle && \text{By distributivity.} \\
&= \sum_k a_k \cdot \overline{A}'_k && \text{By Equation (1.2).} \\
&= \sum_k a_k \cdot (\overline{A}_{11} + r_k \cdot (\overline{A}_{12} + \overline{A}_{21}) + r_k^2 \cdot \overline{A}_{22}) && \text{By verifier output.} \\
&= \Big(\sum_k a_k \Big) \overline{A}_{11} + \Big(\sum_k r_k \cdot a_k \Big) (\overline{A}_{12} + \overline{A}_{21}) + \Big(\sum_k r_k^2 \cdot a_k \Big) \overline{A}_{22} && \text{By distributivity.}
\end{aligned}$$

$= \overline{A}_{11} + \overline{A}_{22}$ By Equation (1.3).

$= \overline{A}$. By verifier check.

Thus, we have that $((G, \overline{A}), A) \in \mathcal{R}_{\mathsf{VC}}(n)$. □

We are still tasked with isolating the base case of the vector commitment proof. Below we specify an *proof of knowledge* for $\mathcal{R}_{\mathsf{VC}}(1)$. A proof of knowledge can be succinctly formalized as a reduction of knowledge that reduces to the relation $\mathcal{R}_\top$ encoding true. A verifier reducing to $\mathcal{R}_\top$ can output true if it accepts and any other string (e.g., false) otherwise.

Definition 1.5 (Proof of Knowledge). Let $\mathcal{R}_\top = \{(\mathsf{true}, \bot)\}$. A proof of knowledge for relation $\mathcal{R}$ is a reduction of knowledge from $\mathcal{R}$ to $\mathcal{R}_\top$.

Construction 1.2 (Base Case). We construct a proof of knowledge for $\mathcal{R}_{\mathsf{VC}}(1)$. Given statement $(G, \overline{A})$ and corresponding witness A, the prover sends A directly to the verifier. The verifier outputs true if $\langle G, A\rangle = \overline{A}$.

We can compose the above reductions to modularly recover the original proof of knowledge for the vector commitment relation. By formalizing each step as a reduction of knowledge, our composition result abstracts away the brunt of the remaining proof effort. In particular, the following corollary holds immediately.

Corollary 1.1 (Vector Commitment proof of Knowledge). Let Π_{VC} denote a reduction of knowledge from $\mathcal{R}_{\mathsf{VC}}(n)$ to $\mathcal{R}_{\mathsf{VC}}(n/2)$ and let Π_{base} denote a proof of knowledge for $\mathcal{R}_{\mathsf{VC}}(1)$. Then

$$\Pi_{\mathsf{base}} \circ \underbrace{\Pi_{\mathsf{VC}} \circ \ldots \circ \Pi_{\mathsf{VC}}}_{\log n \text{ times}}$$

is a proof of knowledge for $\mathcal{R}_{\mathsf{VC}}(n)$ where $n = 2^i$ for $i \in \mathbb{N}$.

1.5 Incrementally Verifiable Computation

Thus far, we have considered proofs of knowledge over linear algebraic statements with recursive structure. More generally, we can consider proofs of knowledge over arbitrary computations with recursive structure. We can prove recursive computations efficiently by designing a proof system that reflects this recursive structure. Such proof systems were originally introduced by Valiant [133] under the name of *incrementally verifiable computation* (IVC). Remarkably, just as we were able to decompose recursive linear algebraic proofs as a sequence of reductions, we can also decompose recursive IVC schemes as a sequence of reductions. By formalizing and proving IVC in the reductions of knowledge framework, we achieve a highly modular construction by stitching together simpler reductions of knowledge.

In more detail, IVC enables a prover to *succinctly* prove the correct execution of $i + 1$ applications of a (non-deterministic) function F by updating a succinct proof of correct

execution of i applications of F. Unlike standard proofs of knowledge, which only consider static computations, recursion allows one to verifiably update the state of a computation over time, enabling stateful computation with dynamic control flow [35, 45, 101, 133]. This has become a pivotal requirement for designing modern distributed and blockchain-based applications, such as verifiable delay functions [34], scalable cryptocurrencies [38, 122], verifiable decentralized storage [47], decentralized private computation [39], and zero-knowledge virtual machines [25, 27].

In particular, IVC is a special type of non-interactive proof of knowledge that demonstrates that n applications of a (non-deterministic) function F on some initial input z_0 results in output z_n. More formally, IVC concerns computational instances of the form (F, n, z_0, z_n). A satisfying witness to this instance is the list of non-deterministic inputs $(\omega_0, \ldots, \omega_{n-1})$ such that for $z'_0 = z_0$ and $z'_{i+1} \leftarrow F(z'_i, \omega_i)$, we have that $z'_n = z_n$.

An IVC scheme is considered knowledge-sound if for any malicious prover $\mathcal{P}^*_{i+1}$ that produces proof π_{i+1} there exists a corresponding *extractor* $\mathcal{E}_i$ that can retrieve a valid proof π_i and witness ω_i. An IVC scheme must additionally satisfy the following two properties: First, given a proof π_i for the instance (F, i, z_0, z_i) and a witness ω_i such that $z_{i+1} \leftarrow F(z_i, \omega_i)$, a prover must be able to efficiently derive an updated proof π_{i+1} for the instance $(F, i+1, z_0, z_{i+1})$ without any additional interaction. Second, the proof π_{i+1} must not grow in size with respect to the proof π_i. These two properties in tandem make IVC a challenging primitive to achieve.

As discussed in Section 1.1.6, Valiant demonstrates how to construct IVC using SNARKs for NP: At each incremental step i, the prover produces a SNARK proving the correct execution of an augmented function F'. The circuit F' runs both the latest iteration of F and a *verification circuit* that checks the SNARK π_i of i iterations of F' by running SNARK verifier (represented as a circuit). Thus, a SNARK π_{i+1} of this circuit attests to $i+1$ iterations of F'. We depict Valiant's approach in Figure 1.1.

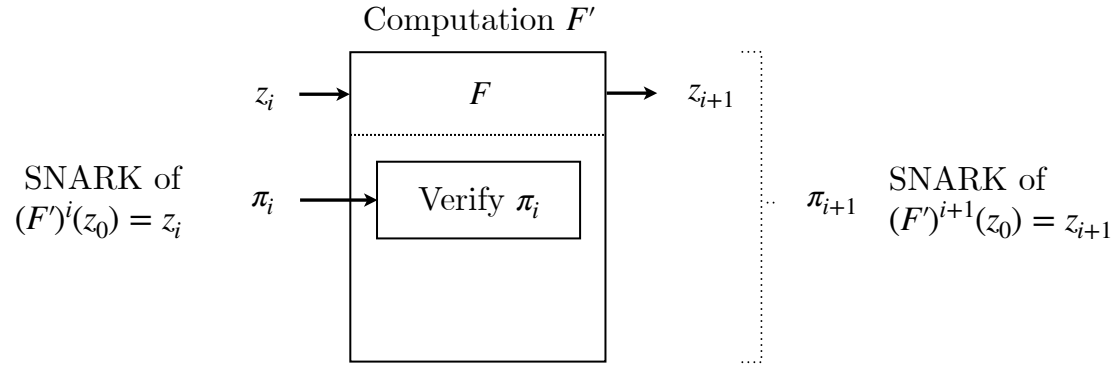


Figure 1.1: Valiant's original IVC construction.

Unfortunately, while Valiant's approach is incredibly insightful in theory, Ben-Sasson et al. [27] show that using standard pairing-based SNARKs [86, 117] would be prohibitively expensive, as it requires representing the SNARK verifier's expensive pairing operations inside the verification circuit. Ben-Sasson et al. suggest using SNARKs that do not rely

on pairings [44, 55, 124, 125], however, the verifiers for these proof systems are asymptotically more expensive, making them inefficient for larger computations. Bowe, Grigg, and Hopwood [38] alongside subsequent works [35, 43] aim to address the inefficiency of SNARK-based IVC with an innovative approach: at each step, the verifier circuit "defers" expensive steps in verifying a SNARK for NP instances (e.g., verifying polynomial evaluation proofs) by accumulating those steps into a single instance that is later checked efficiently. However, these works still require the prover to produce a SNARK at each step and the verifier circuit to partially verify that SNARK. Later, Bünz et al. [45] weaken the requirement for a SNARK by demonstrating that only the portion of the proof that needs to be verifiably deferred needs to be succinct. This enables even more efficient proof machinery in the verifier circuit.

In this section, we overview how to achieve one of the most efficient IVC schemes to date by utilizing reductions of knowledge, namely folding schemes, as opposed to proofs of knowledge. Concretely, our reduction-based approach requires representing just *two group scalar multiplications* inside the verification circuit (each of which takes roughly 1000 constraints to encode) as opposed to three pairing operations using Valiant's SNARK-based approach. This affords significant savings as each pairing is roughly 200× more expensive than a group scalar multiplication as it requires both a more expensive *pairing-friendly* curve and roughly 20× more constraints to encode. We refer to Section 5.1.5 for a concrete cost comparison with the full list of approaches. We proceed as follows: First, in Section 1.5.1, we discuss how each step of IVC can be viewed as a particular type of reduction of knowledge. Next, in Section 1.5.2, assuming the folding scheme for NP from Section 4.2, we sketch a construction for an IVC scheme.

1.5.1 IVC as a Reduction of Knowledge

Incrementally verifiable computation can be understood as a reduction of knowledge from the task of checking knowledge of a proof π_i attesting to the instance (F, i, z_0, z_i) and knowledge of a witness ω_i such that $z_{i+1} = F(z_i, \omega_i)$ to the task of checking a knowledge of a proof π_{i+1} attesting to the instance $(F, i+1, z_0, z_{i+1})$. Of course, the particular relation that specifies a valid proof π_i, denoted Proof, is construction specific. Moreover, the prover must be able to produce a proof π_{i+1} without communicating *at all* with the verifier, a property we refer to as *zero-interactivity*. This is in contrast to *non-interactivity*, which allows the prover to send a single message to the verifier.

To capture statements of the form *"I know ω_i such that $z_{i+1} = F(z_i, \omega_i)$"*. We define the evaluation relation as follows

$$\mathsf{Eval} = \{((F, z_i, z_{i+1}), \omega_i) \mid F(z_i, \omega_i) = z_{i+1}\}.$$

Then, as a first pass, we can define IVC as a zero-interactive reduction of knowledge of type

$$\mathsf{Eval} \times \mathsf{Proof} \to \mathsf{Proof}.$$

for some IVC proof relation Proof with fixed-sized witnesses that attest to IVC instances of the form (F, i, z_0, z_i). That is, IVC reduces the task of checking an instance (F, z_i, z_{i+1})

in Eval and and instance (F, i, z_0, z_i) in Proof to the task of checking a new instance $(F, i+1, z_0, z_{i+1})$ in Proof.

While this is close to our desired definition, several problems remain. In particular, because we want to increment the same function F, and iterate on the previous output z_i, we do not want this reduction to admit inconsistent pairs of instances $((F, z_i, z_{i+1}), (F', i, z_0, z_i'))$ in $\mathsf{Eval} \times \mathsf{Proof}$. Thus, we additionally need to specify that the input instances refer to the same function F and intermediate evaluation z_i. Moreover, we must ensure that, given input instance pair $((F, z_i, z_{i+1}), (F, i, z_0, z_i))$, the output instance is $(F, i+1, z_0, z_{i+1})$.

To address these issues, we introduce *refined reductions of knowledge* (Definition 2.1), which extend the language of reductions of knowledge to capture such constraints in the *type* of the reduction.[1] In particular, to capture the former constraint, we introduce *refined products*, which are characterized by an infix binary relation $\sim$ over pairs of instances (denoted in between the two instances) and are defined as follows

$$\mathcal{R}_1 \overset{\sim}{\times} \mathcal{R}_2 = \{((u_1, u_2), (w_1, w_2)) \in \mathcal{R}_1 \times \mathcal{R}_2 \mid u_1 \sim u_2\}.$$

To capture the latter constraint, we introduce *refined arrows*, which, similarly, are characterized by an infix binary relation $\sim$. In particular, a reduction of knowledge has type $\mathcal{R}_1 \overset{\sim}{\to} \mathcal{R}_2$ if the completeness condition additionally enforces that for every output statement u_2 on input statement u_1, we have that $u_1 \sim u_2$.

With refined reductions of knowledge, we can informally define IVC as follows.

Definition 1.6 (Incrementally Verifiable Computation, Informal). An incrementally verifiable computation scheme is defined by (Proof, Π) where

(i) Proof is a relation with fixed-sized witnesses (referred to as the IVC proof), and instances of the form (F, i, z_0, z_i) for function F, iteration count i, input z_0, and output z_i,

(ii) Π is a zero-interactive reduction of knowledge of type

$$\mathsf{Eval} \overset{\sim_1}{\times} \mathsf{Proof} \overset{\sim_2}{\longrightarrow} \mathsf{Proof}$$

where $\sim_1$ enforces that the input instance pairs refer to the same function F and intermediate input z_i, and $\sim_2$ enforces that the output instance increments the counter i and preserves the same starting input z_0.

Semantically, the above definition enforces that any malicious prover for Π that outputs a satisfying witness for Proof (which, by definition, is the IVC proof π_{i+1}) induces a corresponding extractor which can output a satisfying witness to the input instance of the reduction (which, by definition, is the non-deterministic input ω_i and the prior IVC proof π_i).

We show that these additional refinement relations are naturally preserved under sequential and parallel composition. Given these results, we can see that any protocol that

[1] Refined reductions of knowledge are named after *refinement types* [70], which constrain the output type of a function based on the input.

achieves the above definition can be sequentially composed to produce a reduction that reduces the task of checking some base case instance in Proof, and knowledge of witnesses $\omega_0, \omega_1 \ldots, \omega_{n-1}$ that satisfy instances $(F, z_0, z_1), (F, z_1, z_2), \ldots, (F, z_{n-1}, z_n)$ in Eval to the task of checking a single instance (F, n, z_0, z_n) in Proof.

1.5.2 Constructing IVC

We now overview the IVC construction which we formally develop in Section 4.2 and Section 5.1. For the sake of presentation we overview our IVC construction with a specific folding scheme. However, our formal construction admits any *IVC-compatible* folding scheme (Definition 5.4).

A Folding Scheme for NP

Recall that a folding scheme for relation $\mathcal{R}$ is a reduction of knowledge of type $\mathcal{R} \times \mathcal{R} \to \mathcal{R}$. In Section 4.2 we introduce a folding scheme for a popular NP-complete relation, R1CS [78]. At a high level, this folding scheme is then used to incrementally fold updates to the computation expressed as an R1CS instance into a running proof expressed as an R1CS instance relaxed with additional error terms.

In more detail, an R1CS instance is defined by constraint matrices $(A, B, C) \in \mathbb{F}^{n \times n}$ representing a circuit structure, and a public IO vector x representing inputs and outputs of a circuit. A witness W satisfies an instance $((A, B, C), x)$ if for $Z = (W, x, 1)$

$$AZ \circ BZ = CZ$$

where $\circ$ denotes the Hadamard product (i.e., entry-wise multiplication). An R1CS instance augmented with error terms, which we refer to as a relaxed R1CS instance is additionally defined by scalar u. A witness (W, E) satisfies an instance $((A, B, C), x, u)$ if for $Z = (W, x, u)$

$$AZ \circ BZ = u \cdot CZ + E$$

Note that we can trivially compute a satisfying witness for any relaxed R1CS instance by randomly sampling W and setting E appropriately. This is mitigated by the fact that we only consider relaxed R1CS instances augmented with commitments to the witness updated honestly by the verifier in the instance. Let $\mathsf{R1CS}_{\mathsf{com}}$ refer to the R1CS relation with witness commitments under commitment scheme com. Likewise, let $\mathsf{RR1CS}_{\mathsf{com}'}$ refer to the relaxed R1CS analogue.

Then, as a first pass, we can interpret the folding scheme of Kothapalli et al. as a reduction of knowledge of type

$$\mathsf{R1CS}_{\mathsf{com}} \times \mathsf{RR1CS}_{\mathsf{com}'} \to \mathsf{RR1CS}_{\mathsf{com}'}.$$

At a high level, the folding scheme works by taking a random linear combination of quadratic constraints in both instances and aggregating all the cross terms into the updated error vector. However, a key caveat is that this folding scheme can only fold two instances that refer to the same (A, B, C) matrices. Conversely, this folding scheme ensures that the

output instance maintains the same (A, B, C) matrices, which is critical for achieving IVC. We can capture these properties using the language of refined reductions of knowledge in the lemma below.

Lemma 1.4 (Folding Scheme for R1CS, Informal). There exists a non-interactive folding scheme of type

$$\mathsf{R1CS}_{\mathsf{com}} \overset{\sim_1}{\times} \mathsf{RR1CS}_{\mathsf{com}'} \xrightarrow{\sim_2} \mathsf{RR1CS}_{\mathsf{com}'}$$

in the random oracle model, where $\sim_1$ enforces that both input instances refer to the same (A, B, C) matrices, and $\sim_2$ enforces that the output instance preserves the same (A, B, C) matrices.

Critically, the verifier runtime in our folding scheme for R1CS is dominated by just two group scalar multiplications. This property is what enables us to achieve a remarkably efficient IVC scheme in practice. As with all existing approaches [38, 45, 99, 101], we need a folding scheme in the *plain model* for recursion. Thus, we must instantiate the random oracle with a cryptographic hash function, to heuristically achieve a folding scheme in the plain model.

IVC from a Folding Scheme for NP

We are now ready to construct IVC from the prior folding scheme for R1CS, which is heuristically secure in the plain model. To do so, we will describe the prover's protocol for a single iterative step. That is, given an instance-witness pair $((F, z_i, z_{i+1}), \omega_i) \in \mathsf{Eval}$ and an instance-witness pair $((F, i, z_0, z_i), \pi_i) \in \mathsf{Proof}$ for relation Proof which we will specify shortly, we discuss how a prover can produce an IVC proof π_i such that $((F, i+1, z_0, z_{i+1}), \pi_{i+1}) \in \mathsf{Proof}$.

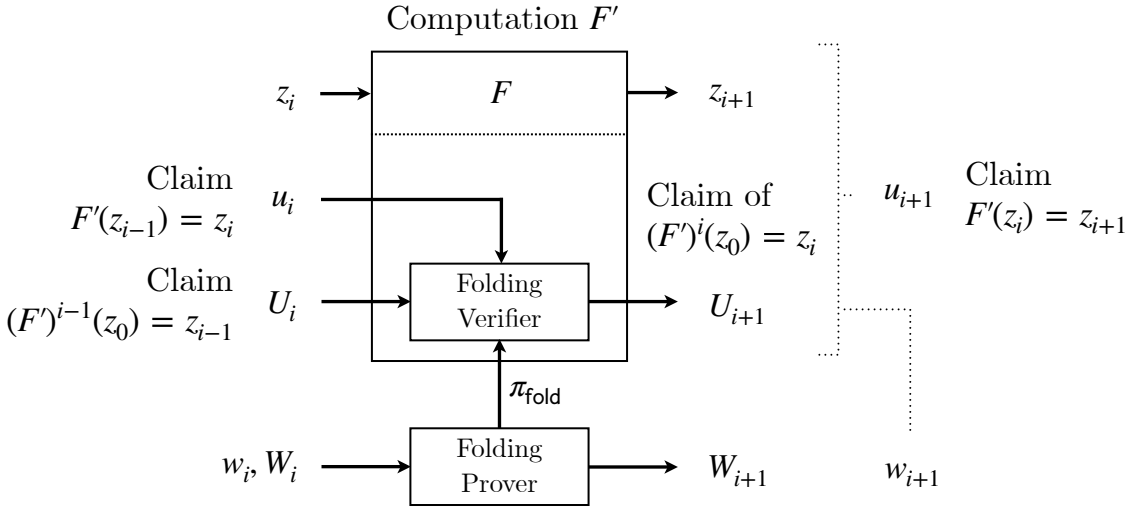


Figure 1.2: Overview of IVC from folding.

At a high level, instead of directly proving knowledge of a satisfying witness ω_i such that $((F, z_i, z_{i+1}), \omega_i) \in \mathsf{Eval}$ in each step, the prover proves knowledge of a satisfying witness to an augmented function F'. The augmented function F', in addition to running F, runs a verifier circuit which uses a folding scheme to help verifiably update the IVC proof π_i. In particular, the verifier circuit additionally takes as input an R1CS instance u_i such that checking this instance implies checking iteration i of F' and a relaxed R1CS instance U_i such that checking this instance implies checking $i-1$ prior iterations of F'. Instead of directly checking either instance, the verifier circuit folds them to produce a new instance U_{i+1} which implies checking i iterations of F'. To claim the correctness of this latest execution of F' the prover produces a new instance u_{i+1}.

We define the IVC proof π_i to contain the relaxed R1CS instance U_i the R1CS instance u_i, and the corresponding witnesses W_i and w_i. Thus, the prover can use parts of π_i as input to F' to produce U_{i+1} and u_{i+1}, and separately compute the corresponding witness W_{i+1} (using the folding prover) and w_{i+1}. These terms together define π_{i+1}. We depict this construction in Figure 1.2.

Crucially, the verifier for our folding scheme for R1CS is dominated by just two group scalar multiplications. This means the verifier circuit in F' can be expressed in merely 20000 gates (Figure 5.2), which is substantially more efficient than existing SNARK-based approaches.

We now sketch the construction in more detail.

Construction 1.3 (IVC, Informal). We first define the augmented function F'. Using F' we define a proof relation Proof. Then, we construct the corresponding zero-interactive reduction of knowledge.

The Augmented Function. The function F' takes as input part of the Proof instance (i, z_0, z_i), and the Eval witness ω_i, the $\mathsf{R1CS}$ instance u_i that claims the correct execution of step i, and the $\mathsf{RR1CS}$ instance U_i that claims the correct execution of $i-1$ prior iterations. F' first computes $z_{i+1} \leftarrow F(z_i, \omega_i)$. As additional bookkeeping, F' does the following.

1. Check that (a succinct commitment to) U_i is contained in the public output of the instance u_i. This enforces that U_i is produced by the prior step.

2. Run the folding scheme verifier to fold u_i into U_i to produce an updated instance U_{i+1}. This ensures that checking U_{i+1} implies checking u_i and U_i while maintaining that U_{i+1} does not grow in size with respect to U_i.

F' produces as public output the new partial instance (i, z_0, z_i), and (a succinct commitment to) the updated relaxed R1CS instance U_{i+1}.

The IVC Proof Relation. Consider a Proof instance (F, i, z_0, z_i). Let the corresponding witness π_i (called the IVC proof) consist of the relaxed R1CS instance U_i, the R1CS instance u_i, and the corresponding witnesses W_i and w_i. The IVC proof π_i is a satisfying witness if (u_i, w_i) and (U_i, W_i) are satisfying $\mathsf{R1CS}$ (respectively $\mathsf{RR1CS}$) instance-witness

pairs with respect to function F', and u_i contains U_i in the public output. We will have that so long as (u_i, w_i) is a satisfying instance-witness pair that contains U_i in the public output, then checking U_i implies checking $i-1$ prior iterations of F'. Then, because checking u_i additionally attests to iteration i of F', we have that checking both u_i and U_i implies checking i prior iterations of F'. Thus, checking π_i implies checking i iterations of the original function F.

The IVC Proof Reduction. Suppose the prover takes as input $((F, z_i, z_{i+1}), \omega_i) \in \mathsf{Eval}$ and $((F, i, z_0, z_i), \pi_i) \in \mathsf{Proof}$. We now discuss how a prover can produce an IVC proof π_i such that $((F, i+1, z_0, z_{i+1}), \pi_{i+1}) \in \mathsf{Proof}$. The core invariant we must maintain is that if checking π_i indeed attests to i iterations of F, then we must have that checking π_{i+1} attests to checking $i+1$ steps while maintaining that π_{i+1} does not grow in size. Indeed, assume that checking π_i implies checking i iterations of F. The prover parses π_i as $((u_i, w_i), (U_i, W_i))$ and computes

$$((i+1, z_0, z_{i+1}), U_{i+1}) \leftarrow F'((i, z_0, z_i), \omega_i, (u_i, U_i))$$

The prover then produces an R1CS instance-witness pair (u_{i+1}, w_{i+1}) which attests to this execution of F'. Now, we have that checking u_{i+1} attests to the following.

1. F produces z_{i+1} on input (z_i, ω_i)
2. The public output of u_i contains U_i.
3. U_{i+1} was computed by folding u_i into U_i, and therefore U_{i+1} can be checked in place of both prior instances.

Therefore, so long as u_{i+1} is satisfying and contains U_{i+1} in the public output, we have that checking U_{i+1} attests to checking i iterations of F'. Then, because u_{i+1} additionally attests to iteration $i+1$ of F', we have that checking both u_{i+1} and U_{i+1} implies checking $i+1$ iterations of F'. This means that checking $\pi_{i+1} = ((u_{i+1}, w_{i+1}), (U_{i+1}, W_{i+1}))$ is sufficient to check $i+1$ iterations of the original function F.

Theorem 1.5 (IVC, Informal). Construction 1.3 is an IVC scheme.

Proof Intuition. Recall that to prove knowledge soundness we must show that for any malicious prover $\mathcal{P}^*$ that outputs a satisfying witness

$$\pi_{i+1} = ((u_{i+1}, w_{i+1}), (U_{i+1}, W_{i+1}))$$

for Proof induces a corresponding extractor $\mathcal{E}$ which can output a satisfying witness (π_i, ω_i) for the previous iteration.

To build such an extractor we first design a malicious prover $\mathcal{P}^*_{\mathsf{FS}}$ for the underlying folding scheme that internally runs the overall prover $\mathcal{P}^*$ *once* to produce a malicious folded instance-witness pair (U_{i+1}, W_{i+1}). By the knowledge-soundness of the underlying folding scheme there exist a corresponding extractor $\mathcal{E}_{\mathsf{FS}}$ which can produce the unfolded instance-witness pairs (u_i, w_i) and (U_i, W_i). We then construct the desired extractor $\mathcal{E}$

which runs $\mathcal{P}^*$ to parse ω_i from the output witness of correct execution w_{i+1} and $\mathcal{E}_{\mathsf{FS}}$ to retrieve (u_i, w_i) and (U_i, W_i). The extractor then outputs the proof $\pi_i = ((u_i, w_i), (U_i, W_i))$ and ω_i. $\square$

Chapter 2

A Theory of Composition

This chapter contains joint work with Bryan Parno [96] and Leah Rosenbloom.

I will argue strongly that composition is the essence of programming.

– Bartosz Milewski,
Category Theory for Programmers

Recall that in contrast to proofs of knowledge, reductions of knowledge are defined over a pair of relations $\mathcal{R}_1$ and $\mathcal{R}_2$. A prover can use a reduction of knowledge to show for some u_1 that it knows w_1 such that $(u_1, w_1) \in \mathcal{R}_1$ contingent on the fact that it knows w_2 for some statement u_2 (derived from its interaction with the verifier) such that $(u_2, w_2) \in \mathcal{R}_2$. We start by intuiting the desired notion of knowledge soundness needed to capture such an interaction, before presenting a formal definition (Definition 2.1). We show that any two reductions of knowledge that respect this definition can be composed sequentially and in parallel (Theorems 2.1 and 2.2). We then observe that a more restricted — but simpler — notion of soundness, known as tree extractability, implies our definition of knowledge soundness (Lemma 2.2). In Chapter 3, we leverage this observation to prove that our reductions of knowledge for linear-algebraic statements are secure.

2.1 Reductions of Knowledge

Intuitively, we would like that if a prover is able to convince a verifier on input u_1 to output some derived statement u_2 such that it knows a corresponding satisfying witness w_2, then it must have known a corresponding satisfying witness w_1 for u_1. We can capture this notion formally by stating that if a malicious prover can output a satisfying witness w_2 for the verifier's output statement u_2, then there must exist a corresponding extractor that can output a satisfying witness w_1 for the verifier's input statement u_1.

While this presents a stand-alone notion of knowledge soundness, we require a more nuanced definition to capture technical difficulties that arise when reasoning about sequential composability. In particular, existing definitions implicitly assume that the environment is provided access to the inputs and outputs of the prover and the verifier, and that some of this material (such as an adversarially chosen statement) is forwarded to the extractor. Unfortunately, when composing such proofs of knowledge, we end up in situations where intermediate inputs expected by the extractor are never exposed to the environment.

Concretely, consider a reduction of knowledge Π_1 with prover $\mathcal{P}_1$, verifier $\mathcal{V}_1$, and extractor $\mathcal{E}_1$, and a second reduction of knowledge Π_2 with corresponding $\mathcal{P}_2$, $\mathcal{V}_2$, and $\mathcal{E}_2$. Ideally, we would want to use $\mathcal{E}_1$ and $\mathcal{E}_2$ in a black-box manner to construct an extractor $\mathcal{E}$ for $\Pi_2 \circ \Pi_1$. A typical knowledge soundness definition would dictate that the statement provided to the verifier is forwarded to the extractor as well. Unfortunately, in the composed setting, the statement u_2 output by $\mathcal{V}_1$ as input to $\mathcal{V}_2$ is never exposed to the environment, and thus it is unclear how $\mathcal{E}$ can simulate the intermediate statement u_2 expected by $\mathcal{E}_2$.

To alleviate this issue, we stipulate an additional requirement that the verifier's output statement can be deterministically recovered from the mutual view of both the prover and verifier. Specifically, the mutual view consists of the public parameters, initial input statement, and interaction transcript. We refer to this property as *public reducibility*, which can be viewed as analogous to the public verifiability property common to most modern proofs of knowledge. With public reducibility, we are afforded sequential composability.

We formally define reductions of knowledge as interactive protocols in the global common reference string model.

Definition 2.1 (Reduction of Knowledge). Consider ternary relations $\mathcal{R}_1$ and $\mathcal{R}_2$ consisting of public parameters, statement, witness tuples. A reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ denoting the generator, the prover, and the verifier respectively with the following interface.

- $\mathcal{G}(\lambda) \to \mathsf{pp}$: Takes security parameter λ. Outputs public parameters pp.
- $\mathcal{P}(\mathsf{pp}, u_1, w_1) \to (u_2, w_2)$: Takes as input public parameters pp, and statement-witness pair (u_1, w_1). Interactively reduces the statement $(\mathsf{pp}, u_1, w_1) \in \mathcal{R}_1$ to a new statement $(\mathsf{pp}, u_2, w_2) \in \mathcal{R}_2$.
- $\mathcal{V}(\mathsf{pp}, u_1) \to u_2$: Takes as input public parameters pp, and statement u_1 associated with $\mathcal{R}_1$. Interactively reduces the task of checking u_1 to the task of checking a new statement u_2 associated with $\mathcal{R}_2$.

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ denote the interaction between $\mathcal{P}$ and $\mathcal{V}$. We treat $\langle \mathcal{P}, \mathcal{V} \rangle$ as a function that takes as input (pp, u_1, w_1) and runs the interaction on prover input (pp, u_1, w_1) and verifier input (pp, u_1). At the end of the interaction, $\langle \mathcal{P}, \mathcal{V} \rangle$ outputs the verifier's statement u_2 and the prover's witness w_2. A reduction of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ satisfies the following conditions.

(i) Completeness: For any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$ and $(u_1, w_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, u_1, w_1) \in \mathcal{R}_1$, we have that the prover's output statement is equal to the verifier's output statement and that

$$(\mathsf{pp}, \langle \mathcal{P}, \mathcal{V} \rangle (\mathsf{pp}, u_1, w_1)) \in \mathcal{R}_2.$$

(ii) Knowledge Soundness: For any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$, there exists an expected polynomial-time extractor $\mathcal{E}$ such that given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$ and $(u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, we have that

$$\Pr[(\mathsf{pp}, u_1, \mathcal{E}(\mathsf{pp}, u_1, \mathsf{st})) \in \mathcal{R}_1] \approx \Pr[(\mathsf{pp}, \langle \mathcal{P}^*, \mathcal{V} \rangle (\mathsf{pp}, u_1, \mathsf{st})) \in \mathcal{R}_2].$$

(iii) Public Reducibility: There exists a deterministic polynomial-time function φ such that for any PPT adversary $\mathcal{A}$ and expected polynomial-time adversary $\mathcal{P}^*$, given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$, $(u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, and $(u_2, w_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle (\mathsf{pp}, u_1, \mathsf{st})$ with interaction transcript tr, we have that $\varphi(\mathsf{pp}, u_1, \mathsf{tr}) = u_2$.

We write $\Pi : \mathcal{R}_1 \to \mathcal{R}_2$ to denote that protocol Π is a reduction of knowledge from relation $\mathcal{R}_1$ to relation $\mathcal{R}_2$.

We define reductions of knowledge in the random oracle model.

Definition 2.2 (Random Oracle Model). A reduction of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ in the random oracle model is defined identically with the exception that all parties are additionally provided oracle access to a random oracle.

Next, we define communication specific properties of reductions of knowledge such as public-coin, succinctness, and non-interactivity.

Definition 2.3 (Public-Coin). A reduction of knowledge is public-coin if the verifier only sends a uniformly random challenge in response to each prover message.

Definition 2.4 (Succinctness). We say a reduction has succinct communication if the communication complexity is sublinear in the size of the input statement and witness. We say a reduction of knowledge has a succinct verification if the verifier time complexity is sublinear in the size of the input statement and witness.

Definition 2.5 (Non-Interactive). A reduction $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is non-interactive if the interaction between $\mathcal{P}$ and $\mathcal{V}$ only consists of messages from the prover. In this setting, we sometimes denote the prover's single message as a proof π that is passed as an additional input to the verifier.

In this work, we introduce a new property, *zero-interactivity*, which is only meaningful in the context of reductions of knowledge (as opposed to proofs of knowledge). In particular, unlike non-interactivity, which captures protocols in which communication consists of a single message from the prover, zero-interactivity captures protocols in which the prover and verifier send no messages *at all*.

This seemingly paradoxical property captures protocols in which the prover's ability to produce a satisfying witness for an instance in the output relation implies knowledge of a witness for a corresponding instance in the input relation, even if there is no communication that indicates this. For example, in the case of IVC, a prover's ability to independently produce an IVC proof for n iterations must *itself* indicate knowledge of an IVC proof for $n-1$ iterations.

Definition 2.6 (Zero-Interactive). A reduction is zero-interactive if the prover and verifier send no messages.

Next, we define two related primitives, proofs of knowledge and Levin reductions (which we use to define several subprotocols), in the reductions of knowledge framework. Recall that a proof of knowledge can be understood as a reduction of knowledge that reduces to $\mathcal{R}_\top = \{\mathsf{true}, \bot\}$, a canonical relation that encodes true. In such a reduction, a verifier can output true if it accepts (for which an honest prover can output the corresponding witness $\bot$) and anything else (e.g., false) otherwise.

Definition 2.7 (Proof of Knowledge). Consider the boolean relation $\mathcal{R}_\top = \{(\mathsf{true}, \bot)\}$. A proof of knowledge for relation $\mathcal{R}$ is a reduction of knowledge of type $\mathcal{R} \to \mathcal{R}_\top$.

Recall that a Levin reduction is a Karp reduction that maps witnesses as well as instances between relations. Formally, for relations $\mathcal{R}_1$ and $\mathcal{R}_2$, a Levin reduction from $\mathcal{R}_1$ to $\mathcal{R}_2$ consists of deterministic polynomial-time functions p, v, and e such that the following conditions hold.

1. Function v is a Karp reduction from $\mathcal{L}(\mathcal{R}_1)$ (i.e., the language associated with relation $\mathcal{R}_1$) to $\mathcal{L}(\mathcal{R}_2)$.
2. If $(u_1, w_1) \in \mathcal{R}_1$, then $(v(u_1), p(u_1, w_1)) \in \mathcal{R}_2$.
3. If $(v(u_1), w_2) \in \mathcal{R}_2$, then $(u_1, e(u_1, w_2)) \in \mathcal{R}_1$.

From this definition, we can see that p, v, and e correspond to the prover, verifier, and extractor of a reduction of knowledge. In particular, a Levin reduction can be interpreted as a special type of reduction of knowledge in which there is no common reference string, the prover and verifier are deterministic, and there is no communication.

Definition 2.8 (Levin Reduction). A reduction of knowledge $\Pi : \mathcal{R}_1 \to \mathcal{R}_2$ is a Levin reduction if the generator outputs $\bot$, it is deterministic (i.e., the prover and verifier do not use their random tapes), and is zero-interactive.

2.2 Composing Reductions of Knowledge

We now prove sequential and parallel composition theorems for reductions of knowledge. This allows us to construct complex proofs of knowledge by stitching together simpler reductions sequentially and in parallel. In the case of sequential composition, much like recursive composition techniques [30, 45, 101, 133], each composition step induces a polynomial blowup in the corresponding extractor. Thus, sequential composition cannot be used more than a constant number of times without additional computational assumptions.[1] Our parallel composition operator is not parallel in the sense that both protocols are being run at the same time, but rather parallel in the sense that the composed protocol takes incoming instance-witness pairs in parallel and produces outgoing instance-witness pairs in parallel.

Theorem 2.1 (Sequential Composition). Consider ternary relations $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$. For reductions of knowledge $\Pi_1 = (\mathcal{G}, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 = (\mathcal{G}, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_2 \to \mathcal{R}_3$, we have that $\Pi_2 \circ \Pi_1 = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_3$ where

$$\begin{aligned} \mathcal{P}(\mathsf{pp}, u_1, w_1) &= \mathcal{P}_2(\mathsf{pp}, \mathcal{P}_1(\mathsf{pp}, u_1, w_1)) \\ \mathcal{V}(\mathsf{pp}, u_1) &= \mathcal{V}_2(\mathsf{pp}, \mathcal{V}_1(\mathsf{pp}, u_1, w_1)). \end{aligned}$$

Proof Intuition. Completeness and public reducibility follow by observation. As for knowledge soundness, assume there exists an adversarial prover $\mathcal{P}^*$ for Π that succeeds in producing an accepting witness w_3 with non-negligible probability. Using the second half of $\mathcal{P}^*$ (i.e., the part that interacts with $\mathcal{V}_2$), we can construct an adversary $\mathcal{P}_2^{**}$ for Π_2 that succeeds in producing an accepting witness w_3 with the same probability. By the knowledge soundness of Π_2, this implies an extractor $\mathcal{E}_2$ that succeeds in producing an intermediate witness w_2 with nearly the same probability. We can then leverage $\mathcal{E}_2$ to construct an adversary $\mathcal{P}_1^{**}$ for Π_1 that succeeds in producing an accepting witness w_2 with nearly the same probability. In particular, $\mathcal{P}_1^{**}$ first runs the first half of $\mathcal{P}^*$ and then runs extractor $\mathcal{E}_2$ on the intermediate statement u_2 (derived by the public reducibility of Π_1) and the intermediate state of $\mathcal{P}^*$ to produce the output w_2. By the knowledge soundness of Π_1, this implies the desired extractor $\mathcal{E}_1$ that succeeds in producing the witness w_1 with nearly the same probability. □

Proof. Completeness and public reducibility follow by observation. As for knowledge soundness, consider arbitrary expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$. Let $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$ and let $(u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$. Suppose that

$$\Pr[(\mathsf{pp}, \langle \mathcal{P}^*, \mathcal{V} \rangle(\mathsf{pp}, u_1, \mathsf{st})) \in \mathcal{R}_3] = \epsilon. \tag{2.1}$$

We must construct an expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr[(\mathsf{pp}, u_1, \mathcal{E}(\mathsf{pp}, u_1, \mathsf{st})) \in \mathcal{R}_1] = \epsilon - \mathsf{negl}(\lambda).$$

[1] We recommend Bitansky et al. [30, Remark 6.3] for details on such assumptions.

At a high level, we proceed as follows: We leverage $\mathcal{A}$ and $\mathcal{P}^*$ to construct expected polynomial-time adversaries $\mathcal{A}_2$ and $\mathcal{P}_2^{**}$ for protocol Π_2 that succeed in producing a satisfying witness w_3 with probability ϵ. By the knowledge soundness of Π_2, this implies an expected polynomial-time extractor $\mathcal{E}_2$ that succeeds in producing a satisfying intermediate witness w_2 with probability $\epsilon - \mathsf{negl}(\lambda)$. We then leverage $\mathcal{E}_2$ (in addition to $\mathcal{A}$ and $\mathcal{P}^*$) to construct expected polynomial-time adversaries $\mathcal{A}_1$ and $\mathcal{P}_1^{**}$ for protocol Π_1 that succeed in producing a satisfying witness w_2 with probability $\epsilon - \mathsf{negl}(\lambda)$. This implies an expected polynomial-time extractor $\mathcal{E}_1$ that succeeds in producing witness w_1 with probability $\epsilon - \mathsf{negl}(\lambda)$.

Indeed, we start by constructing adversaries $\mathcal{A}_2$ and $\mathcal{P}_2^{**}$ for Π_2. By construction, we have that $\mathcal{V}$ first runs $\mathcal{V}_1$ to produce an intermediate statement u_2 and then runs $\mathcal{V}_2$ with input u_2. As such, we have that $\mathcal{P}^*$ first runs some $\mathcal{P}_1^*$ and then runs some $\mathcal{P}_2^*$ such that $\mathcal{P}_1^*$ interacts with $\mathcal{V}_1$ and then passes some state to $\mathcal{P}_2^*$ which interacts with $\mathcal{V}_2$ before the two parties collectively produce the output (u_3, w_3). Therefore, for $(u_2, \mathsf{st}_2) \leftarrow \langle \mathcal{P}_1^*, \mathcal{V}_1 \rangle(\mathsf{pp}, u_1, \mathsf{st})$, by Equation (2.1) we have that

$$\Pr[(\mathsf{pp}, \langle \mathcal{P}_2^*, \mathcal{V}_2 \rangle(\mathsf{pp}, u_2, \mathsf{st}_2)) \in \mathcal{R}_3] = \epsilon. \tag{2.2}$$

Thus, we define $\mathcal{A}_2$ and $\mathcal{P}_2^{**}$ as follows.

$\underline{\mathcal{A}_2(\mathsf{pp}) \to (u_2, \mathsf{st}_2)}$:

1. Compute $(u_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$.
2. Compute $(u_2, \mathsf{st}_2) \leftarrow \langle \mathcal{P}_1^*, \mathcal{V}_1 \rangle(\mathsf{pp}, u_1, \mathsf{st}_1)$.
3. Output (u_2, st_2).

$\underline{\mathcal{P}_2^{**}(\mathsf{pp}, u_2, \mathsf{st}_2) \to (u_3, w_3)}$:

1. Run $\mathcal{P}_2^*(\mathsf{pp}, u_2, \mathsf{st}_2)$, which, at the end of interaction, produces output (u_3, w_3).
2. Output (u_3, w_3).

Suppose now that $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$ and $(u_2, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{pp})$. By construction, $\mathcal{A}_2$ produces the same distribution of outputs as $\langle \mathcal{P}_1^*, \mathcal{V}_1 \rangle(\mathsf{pp}, u_1, \mathsf{st})$. Then, because $\mathcal{P}_2^{**}$ runs $\mathcal{P}_2^*$, by Equation (2.2), we have that

$$\Pr[(\mathsf{pp}, \langle \mathcal{P}_2^{**}, \mathcal{V}_2 \rangle(\mathsf{pp}, u_2, \mathsf{st}_2)) \in \mathcal{R}_3] = \epsilon. \tag{2.3}$$

Then, by the knowledge soundness of Π_2, Equation (2.3) implies that there exists expected polynomial-time extractor $\mathcal{E}_2$ such that

$$\Pr[(\mathsf{pp}, u_2, \mathcal{E}_2(\mathsf{pp}, u_2, \mathsf{st}_2)) \in \mathcal{R}_2] = \epsilon - \mathsf{negl}(\lambda). \tag{2.4}$$

We leverage $\mathcal{E}_2$ to construct adversaries $\mathcal{A}_1$ and $\mathcal{P}_1^{**}$ for Π_1 as follows.

$\underline{\mathcal{A}_1(\mathsf{pp}) \to (u_1, \mathsf{st}_1)}$:

1. Compute and output $(u_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$.

$\underline{\mathcal{P}_1^{**}(\mathsf{pp}, u_1, \mathsf{st}_1) \to (u_3, w_3)}$:

1. Run $\mathcal{P}_1^*(\mathsf{pp}, u_1, \mathsf{st}_1)$, which, at the end of interaction produces intermediate state st_2. Record the corresponding interaction transcript as tr.
2. Use the deterministic polynomial-time function φ guaranteed by the public reducibility property of Π_1 to compute $u_2 \leftarrow \varphi(\mathsf{pp}, u_1, \mathsf{tr})$.
3. Compute $w_2 \leftarrow \mathcal{E}_2(\mathsf{pp}, u_2, \mathsf{st}_2)$.
4. Output (u_2, w_2).

Suppose now that $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$ and $(u_1, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathsf{pp})$. By construction of $\mathcal{A}_1$ and $\mathcal{P}_1^{**}$, the extractor $\mathcal{E}_2$ run by $\mathcal{P}_1^{**}$ is provided the same distribution of inputs as the extractor $\mathcal{E}_2$ in Equation (2.4). Thus, by Equation (2.4), we have that

$$\Pr[(\mathsf{pp}, \langle \mathcal{P}_1^{**}, \mathcal{V}_1 \rangle(\mathsf{pp}, u_1, \mathsf{st}_1)) \in \mathcal{R}_2] = \epsilon - \mathsf{negl}(\lambda). \tag{2.5}$$

Then, by the knowledge soundness of Π_1, Equation (2.5) implies that there exists expected polynomial-time extractor $\mathcal{E}_1$ such that

$$\Pr[(u_1, \mathcal{E}_1(\mathsf{pp}, u_1, \mathsf{st}_1)) \in \mathcal{R}_1] = \epsilon - \mathsf{negl}(\lambda). \tag{2.6}$$

Thus, we can construct the desired extractor $\mathcal{E}$ as follows.

$\underline{\mathcal{E}(\mathsf{pp}, u_1, \mathsf{st}) \to w_1}$:

1. Compute and output $w_1 \leftarrow \mathcal{E}_1(\mathsf{pp}, u_1, \mathsf{st})$.

Suppose now that $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$ and $(u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$. By construction of $\mathcal{E}$, the extractor $\mathcal{E}_1$ run by $\mathcal{E}$ is provided the same distribution of inputs as the extractor $\mathcal{E}_1$ in Equation (2.6). Thus, by Equation (2.6), we have that

$$\Pr[(\mathsf{pp}, u_1, \mathcal{E}(\mathsf{pp}, u_1, \mathsf{st})) \in \mathcal{R}_1] = \epsilon - \mathsf{negl}(\lambda).$$

□

Definition 2.9 (Relation Pair). Consider ternary relations $\mathcal{R}_1$ and $\mathcal{R}_2$ over public parameters, statement, witness tuples. We define the relation

$$\mathcal{R}_1 \times \mathcal{R}_2 = \{(\mathsf{pp}, (u_1, u_2), (w_1, w_2)) \mid (\mathsf{pp}, u_1, w_1) \in \mathcal{R}_1, (\mathsf{pp}, u_2, w_2) \in \mathcal{R}_2\}.$$

We let $\mathcal{R}^\ell$ denote $\mathcal{R} \times \ldots \times \mathcal{R}$ for ℓ times.

Theorem 2.2 (Parallel Composition). Consider ternary relations $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_4$. For reductions of knowledge $\Pi_1 = (\mathcal{G}, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 = (\mathcal{G}, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_3 \to \mathcal{R}_4$, we have that $\Pi_1 \times \Pi_2 = (\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a reduction of knowledge from $\mathcal{R}_1 \times \mathcal{R}_3$ to $\mathcal{R}_2 \times \mathcal{R}_4$ where

$$\mathcal{P}(\mathsf{pp}, (u_1, u_3), (w_1, w_3)) = (\mathcal{P}_1(\mathsf{pp}, u_1, w_1), \mathcal{P}_2(\mathsf{pp}, u_3, w_3))$$
$$\mathcal{V}(\mathsf{pp}, (u_1, u_3)) = (\mathcal{V}_1(\mathsf{pp}, u_1), \mathcal{V}_2(\mathsf{pp}, u_3)).$$

Proof Intuition. For $i \in \{1, 2\}$, we leverage a malicious prover $\mathcal{P}^*$ for Π to construct a prover $\mathcal{P}_i^*$ for protocol Π_i that succeeds in producing a satisfying output witness with the same probability. By the knowledge soundness of Π_i, this implies a corresponding extractor $\mathcal{E}_i$ that succeeds in producing a satisfying input witness with nearly the same probability. These extractors imply the desired extractor $\mathcal{E}$. □

Proof. Completeness and public reducibility follow by observation. As for knowledge soundness, consider arbitrary expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$. Let $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$ and let $(u_1, u_3) \leftarrow \mathcal{A}(\mathsf{pp})$. Suppose that

$$\Pr[(\mathsf{pp}, \langle \mathcal{P}^*, \mathcal{V} \rangle(\mathsf{pp}, (u_1, u_3), \mathsf{st})) \in \mathcal{R}_2 \times \mathcal{R}_4] = \epsilon.$$

We must construct expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr[(\mathsf{pp}, (u_1, u_3), \mathcal{E}(\mathsf{pp}, (u_1, u_3), \mathsf{st})) \in \mathcal{R}_1 \times \mathcal{R}_3] = \epsilon - \mathsf{negl}(\lambda).$$

At a high level, we proceed as follows: For $i \in \{1, 2\}$, we leverage $\mathcal{A}$ and $\mathcal{P}^*$ to construct expected polynomial time adversaries $\mathcal{A}_i$ and $\mathcal{P}_i^*$ for protocol Π_i that succeeds in producing a satisfying output witness with probability ϵ. By the knowledge soundness of Π_i, this implies an expected polynomial-time extractor $\mathcal{E}_i$ that succeeds in producing a satisfying input witness with probability $\epsilon - \mathsf{negl}(\lambda)$. Together these extractors imply the desired extractor $\mathcal{E}$.

Indeed, we construct adversaries $\mathcal{A}_1$ and $\mathcal{P}_1^*$ as follows.

$\underline{\mathcal{A}_1(\mathsf{pp}) \to (u_1, \mathsf{st}_1)}$:

1. Compute $((u_1, u_3), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$.
2. Output statement u_1 and state $\mathsf{st}_1 \leftarrow (u_3, \mathsf{st})$.

$\underline{\mathcal{P}_1(\mathsf{pp}, u_1, \mathsf{st}_1) \to (u_2, w_2)}$:

1. Parse st_1 as (u_3, st).
2. Run $\mathcal{P}^*(\mathsf{pp}, (u_1, u_3), \mathsf{st})$. At the end of interaction, $\mathcal{P}^*$ produces statement pair (u_2, u_4) and corresponding witness pair (w_2, w_4).
3. Output (u_2, w_2).

Suppose now that $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$ and $(u_1, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathsf{pp})$. By construction, we have that

$$\Pr[(\mathsf{pp}, \langle \mathcal{P}_1^*, \mathcal{V}_1 \rangle(\mathsf{pp}, u_1, \mathsf{st}_1)) \in \mathcal{R}_2] = \epsilon. \tag{2.7}$$

Then, by the knowledge soundness of Π_1, Equation 2.7 implies that there exists expected polynomial-time extractor $\mathcal{E}_1$ such that

$$\Pr[(\mathsf{pp}, u_1, \mathcal{E}_1(\mathsf{pp}, u_1, \mathsf{st}_1)) \in \mathcal{R}_1] = \epsilon - \mathsf{negl}(\lambda). \tag{2.8}$$

Similarly, we can design adversaries $\mathcal{A}_2$ and $\mathcal{P}_2^*$ for Π_2 such that $\mathcal{P}_2^*$ succeeds with probability ϵ. Then, by the knowledge soundness of Π_2, there exists expected polynomial-time extractor $\mathcal{E}_2$ such that for $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$ and $(u_3, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{pp})$ we have that

$$\Pr[(u_3, \mathcal{E}_2(\mathsf{pp}, u_3, \mathsf{st}_2)) \in \mathcal{R}_3] = \epsilon - \mathsf{negl}(\lambda). \tag{2.9}$$

Thus, we can construct the desired extractor $\mathcal{E}$ as follows.

$\underline{\mathcal{E}(\mathsf{pp}, (u_1, u_3), \mathsf{st}) \to (w_1, w_3)}$:

1. Compute $w_1 \leftarrow \mathcal{E}_1(\mathsf{pp}, u_1, (\mathsf{pp}, u_3, \mathsf{st}))$.
2. Compute $w_3 \leftarrow \mathcal{E}_2(\mathsf{pp}, u_2, (\mathsf{pp}, u_1, \mathsf{st}))$.
3. Output (w_1, w_3).

Suppose now that $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$ and $((u_1, u_3), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$. By construction of $\mathcal{E}$ the extractors $\mathcal{E}_1$ and $\mathcal{E}_2$ are provided the same distribution of inputs as the extractors in Equations (2.8) and (2.9) respectively. Therefore, by Equations (2.8) and (2.9), we have that

$$\Pr[((u_1, u_3), \mathcal{E}(\mathsf{pp}, (u_1, u_3), \mathsf{st})) \in \mathcal{R}_1 \times \mathcal{R}_3] = \epsilon - \mathsf{negl}(\lambda).$$

□

Lemma 2.1 (Closure of Interactivity). Public-coin, succinctness, non-interactivity and zero-interactivity are closed under sequential and parallel composition.

2.3 Knowledge Soundness from Tree Extraction

When proving constructions secure, reasoning about knowledge soundness directly is typically cumbersome. To alleviate this issue, prior work [36] observes that most protocols are algebraic: The corresponding extractor typically runs the malicious prover multiple times with refreshed verifier randomness to retrieve accepting transcripts, which can be interpolated to retrieve the witness. Leveraging this insight, Bootle et al. [36] provide a general extraction lemma, which states that to prove knowledge soundness for algebraic protocols, it is sufficient to show that there exists an extractor that can produce a satisfying witness when provided a *tree of accepting transcripts* with refreshed verifier randomness at each layer. This proof technique has been adapted to various settings [37, 42, 45, 101], and we similarly provide the corresponding lemma for reductions of knowledge.

Definition 2.10 (Tree of Transcripts). Consider an m-round public-coin interactive protocol $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ that satisfies the interface described in Definition 2.1. A $(n_1, \ldots, n_m)$-tree of accepting transcripts for statement u_1 is a tree of depth m where each vertex at layer i has n_i outgoing edges such that (1) each vertex in layer $i \in [m]$ is labeled with a prover message for round i; (2) each outgoing edge from layer $i \in [m]$ is labeled with a different choice of verifier randomness for round i; (3) each leaf is labeled with an accepting statement-witness pair output by the prover and verifier corresponding to the interaction along the path.

Lemma 2.2 (Tree Extraction [37]). Consider an m-round public-coin interactive protocol $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ that satisfies the interface described in Definition 2.1 and satisfies completeness. Then $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is a reduction of knowledge if there exists a PPT extractor χ that, for all instances u_1, outputs a satisfying witness w_1 with probability $1 - \mathsf{negl}(\lambda)$, given an $(n_1, \ldots, n_m)$-tree of accepting transcripts for u_1 where the verifier's randomness is sampled from space Q such that $|Q| = O(2^\lambda)$, and $\prod_i n_i = \mathsf{poly}(\lambda)$.

Proof Intuition. Our proof closely follows that of Bootle et al. [36]. At a high level, we construct an expected polynomial-time extractor $\mathcal{E}$ that repeatedly runs the malicious prover $\mathcal{P}^*$ and collects corresponding accepting transcripts and associated output statement-witness pairs. The extractor then passes these collected transcripts to χ which retrieves the desired witness by assumption. □

Proof. Public reducibility follows from the completeness and public-coin properties. In particular, we have that there exists a deterministic function $\mathcal{V}'$, presumably run by the verifier at the end of interaction, which produces the verifier's output statement u_2 on input public parameters pp, statement u_1, interaction transcript tr, and private randomness $r_\mathcal{V}$ (i.e., randomness not sent in the transcript). Likewise, there exists a deterministic function $\mathcal{P}'$ which produces the prover's output statement u_2 on input public parameters pp statement u_1, witness w_1, interaction transcript tr, and private randomness $r_\mathcal{P}$. Consider arbitrary expected polynomial-time adversary $\mathcal{A}$. Let $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$, $(u_1, w_1) \leftarrow \mathcal{A}(\mathsf{pp})$, and suppose the interaction $\langle \mathcal{P}, \mathcal{V} \rangle(\mathsf{pp}, u_1, w_1)$ produces corresponding transcript tr with private prover randomness $r_\mathcal{P}$. By the public-coin property, we have that tr is independent of $r_\mathcal{V}$. Then, by the completeness property, for all $r_\mathcal{V}$ we have that

$$\mathcal{V}'(\mathsf{pp}, u_1, \mathsf{tr}, r_\mathcal{V}) = \mathcal{P}'(\mathsf{pp}, u_1, w_1, \mathsf{tr}, r_\mathcal{P}).$$

Thus, we have that $\varphi(\mathsf{pp}, u_1, \mathsf{tr}) = \mathcal{V}'(\mathsf{pp}, u_1, \mathsf{tr}, \perp)$ produces the prover and verifier's output statement.

As for knowledge soundness, our proof follows that of [36]. At a high level, we construct an expected polynomial-time extractor $\mathcal{E}$ that repeatedly runs the malicious prover $\mathcal{P}^*$ and collects corresponding accepting transcripts and associated output statement-witness pairs. The extractor then passes these collected transcripts to χ which retrieves the desired witness by assumption.

In more detail, consider an m-round public-coin interactive protocol $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ that presumably reduces from $\mathcal{R}_1$ to $\mathcal{R}_2$. Suppose there exists adversary $\mathcal{P}^*$ that succeeds with probability ϵ and extractor χ that succeeds with probability $1 - \mathsf{negl}(\lambda)$. We are tasked with constructing extractor $\mathcal{E}$ that succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$. Indeed, let $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$, and $(u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$. We construct extractor $\mathcal{E}$ as follows.

$\underline{\mathcal{E}(\mathsf{pp}) \rightarrow w_1}$:

1. Compute $\mathsf{tree} \leftarrow \mathsf{TreeGen}(1)$.
2. If tree is not a valid $(n_1, \ldots, n_m)$-tree (i.e., there are collisions in the verifier's randomness), output $\perp$.

3. Output $w_1 \leftarrow \mathcal{X}(\mathsf{tree})$

where we define function $\mathsf{TreeGen}$ as follows.

$\underline{\mathsf{TreeGen}(i) \rightarrow \mathsf{tree}}$:

1. Sample fresh verifier randomness r_i for round i.
2. Compute the interaction $\langle \mathcal{P}^*, \mathcal{V} \rangle(\mathsf{pp}, u_1, \mathsf{st})$ up to round i.
3. If $i = m$, then the interaction is complete. Let tr be the corresponding transcript. Let (u_2, w_2) be the verifier's output statement and the prover's corresponding output witness. If $(u_2, w_2) \in \mathcal{R}_2$, output $\{(\mathsf{tr}, (u_2, w_2))\}$. Otherwise output $\perp$.
4. Compute $\mathsf{tree} \leftarrow \mathsf{TreeGen}(i+1)$. If $\mathsf{tree} = \perp$, then return $\perp$.
5. Repeatedly run $\mathsf{TreeGen}(i+1)$ to retrieve $n_{i+1} - 1$ additional accepting subtrees. Append all results to tree and output tree.

We now argue that $\mathcal{E}$ succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$. Let E_{nempty} denote the event that $\mathsf{TreeGen}$ outputs $\mathsf{tree} \neq \perp$ in less than T time steps (we will specify T later). Given E_{nempty}, let E_{valid} denote the event that tree is valid (i.e., there are no collisions in the verifer's randomness). Given E_{nempty} and E_{valid}, let E_{ext} denote the event that χ successfully extracts a valid witness with input tree. Then, we have that $\mathcal{E}$ succeeds with probability

$$P_{\mathcal{E}} = \Pr[E_{\mathsf{nempty}}] \cdot \Pr[E_{\mathsf{valid}}] \cdot \Pr[E_{\mathsf{ext}}].$$

We will compute each of these probabilities.

To compute $\Pr[E_{\mathsf{nempty}}]$ we observe that $\mathsf{TreeGen}(1)$ succeeds so long as its first call to $\mathsf{TreeGen}(2)$ succeeds. Likewise, $\mathsf{TreeGen}(2)$ succeeds so long as its first call to $\mathsf{TreeGen}(3)$ succeeds. Chaining these assertions, we have that $\mathsf{TreeGen}(1)$ succeeds if $\mathsf{TreeGen}(m)$ succeeds, which happens with with probability ϵ. Moreover, the expected number of times $\mathsf{TreeGen}(i)$ calls $\mathsf{TreeGen}(i+1)$ is

$$\begin{aligned}
&1 + \Pr[\text{First call to } \mathsf{TreeGen}(i+1) \text{ succeeds}] \cdot \frac{n_{i+1} - 1}{\Pr[\mathsf{TreeGen}(i+1) \text{ succeeds}]} \\
&= 1 + \epsilon \cdot \frac{n_{i+1} - 1}{\epsilon} \\
&= n_{i+1}.
\end{aligned}$$

Hence, the total runtime of $\mathsf{TreeGen}(1)$ is expected to be $t = O(\prod_{i=1}^{m} n_i)$ which is bounded by $\mathsf{poly}(\lambda)$ by assumption. Then, by Markov's inequality, we have that $\mathsf{TreeGen}(1)$ runs for time $T > t$ with probability $\frac{t}{T}$. Thus, we have that

$$\Pr[E_{\mathsf{nempty}}] = \left(1 - \frac{t}{T}\right) \cdot \epsilon$$

Given E_{nempty}, we have that $\mathsf{TreeGen}(1)$ runs for at most T steps. But this means that there are at most T random challenges produced for the verifier implying that the probability of collision is at most $\frac{T^2}{|Q|}$. Thus, we have

$$\Pr[E_{\mathsf{valid}}] = 1 - \frac{T^2}{|Q|}.$$

Finally, given E_{nempty} and E_{valid}, we have that $\Pr[E_{\mathsf{ext}}]$ is $1 - \mathsf{negl}(\lambda)$ by assumption. Now, setting $T = \sqrt[3]{|Q|}$, we have

$$\begin{aligned}
P_{\mathcal{E}} &= \Pr[E_{\mathsf{nempty}}] \cdot \Pr[E_{\mathsf{valid}}] \cdot \Pr[E_{\mathsf{ext}}] \\
&= \left(1 - \frac{t}{T}\right) \cdot \epsilon \cdot \left(1 - \frac{T^2}{|Q|}\right) \cdot (1 - \mathsf{negl}(\lambda)) \\
&= \left(1 - \frac{t}{\sqrt[3]{|Q|}}\right) \cdot \epsilon \cdot \left(1 - \frac{1}{\sqrt[3]{|Q|}}\right) \cdot (1 - \mathsf{negl}(\lambda)) \\
&= \epsilon - \mathsf{negl}(\lambda).
\end{aligned}$$

Thus, we have that $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ satisfies knowledge soundness. □

2.4 Structured Reductions of Knowledge

In many cases, we would like to add a preprocessing stage, where given the public parameters and a reusable structure (e.g., a circuit to be used over multiple inputs), an *encoder* can produce a prover key and a (succinct) verifier key. As we will see throughout our development, this is critical for achieving a succinct verifier. Below, we define *structured* reductions of knowledge, which augments the original definition with the notion of a structure and an encoder algorithm. In particular, input and output instance-witness pairs are enforced against a fixed structure which is carried (implicitly) through the reduction.

Definition 2.11 (Structured Reduction of Knowledge). Consider relations $\mathcal{R}_1$, and $\mathcal{R}_2$, over public parameters, structure, instance, and witness tuples. A structured reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic algorithm $\mathcal{K}$, denoting the generator, the prover, the verifier and the encoder respectively with the following interface.

- $\mathcal{G}(\lambda, n) \to \mathsf{pp}$: Takes as input security parameter λ and size parameters n. Outputs public parameters pp.
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}) \to (\mathsf{pk}, \mathsf{vk})$: Takes as input public parameters pp and structure s. Outputs prover key pk and verifier key vk
- $\mathcal{P}(\mathsf{pk}, u_1, w_1) \to (u_2, w_2)$: Takes as input public parameters pp, and statement-witness pair (u_1, w_1). Interactively reduces the statement $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$ to a new statement $(\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2$

- $\mathcal{V}(\mathsf{pk}, u_1) \to u_2$: Takes as input public parameters pp, and statement u_1 associated with $\mathcal{R}_1$. Interactively reduces the task of checking u_1 to the task of checking a new statement u_2 associated with $\mathcal{R}_2$

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ denote the interaction between $\mathcal{P}$ and $\mathcal{V}$. We treat $\langle \mathcal{P}, \mathcal{V} \rangle$ as a function that takes as input $((\mathsf{pk}, \mathsf{vk}), u_1, w_1)$ and runs the interaction on prover input (pk, u_1, w_1) and verifier input (pp, u_1). At the end of the interaction, $\langle \mathcal{P}, \mathcal{V} \rangle$ outputs the verifier's statement u_2 and the prover's witness w_2. A reduction of knowledge $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies the following conditions.

(i) Completeness: For any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$, $(\mathsf{s}, u_1, w_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$ and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$ we have that the prover's output statement is equal to the verifier's output statement u_2, and that

$$(\mathsf{pp}, \mathsf{s}, \langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, w_1)) \in \mathcal{R}_2.$$

(ii) Knowledge Soundness: For any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$, there exists an expected polynomial-time extractor $\mathcal{E}$ such that given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$, $(\mathsf{s}, u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$, we have that

$$\Pr[(\mathsf{pp}, \mathsf{s}, u_1, \mathcal{E}(\mathsf{pp}, u_1, \mathsf{st})) \in \mathcal{R}_1] \approx \Pr[(\mathsf{pp}, \mathsf{s}, \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})) \in \mathcal{R}_2].$$

(iii) Public Reducibility: There exists a deterministic polynomial-time function φ such that for any PPT adversary $\mathcal{A}$ and expected polynomial-time adversary $\mathcal{P}^*$, given

$$\begin{aligned} \mathsf{pp} &\leftarrow \mathcal{G}(\lambda, n), \\ (\mathsf{s}, u_1, \mathsf{st}) &\leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) &\leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}), \end{aligned}$$

and given $(u_2, w_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})$ with interaction transcript tr, we have that $\varphi(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{tr}) = u_2$.

We correspondingly define the structured variant of a relation product, where the product additionally enforces that both input instances are with respect to the same structure.

Definition 2.12 (Structured Relation Product). For relations $\mathcal{R}_1$ and $\mathcal{R}_2$, over public parameter, structure, instance, and witness pairs we define the structured product relation as follows.

$$\mathcal{R}_1 \times \mathcal{R}_2 = \left\{ (\mathsf{pp}, \mathsf{s}, (u_1, u_2), (w_1, w_2)) \;\middle|\; \begin{array}{l} (\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1, \\ (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2, \end{array} \right\}.$$

Moreover, we present the corresponding sequential and parallel composition lemmas, which follow immediately from Theorems 2.1 and 2.2.

Lemma 2.3 (Sequential Composition). For reductions $\Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_2 \to \mathcal{R}_3$, we have that $\Pi_2 \circ \Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \to \mathcal{R}_3$ where

$$\mathcal{P}(\mathsf{pk}, u_1, w_1) = \mathcal{P}_2(\mathsf{pk}, \mathcal{P}_1(\mathsf{pk}, u_1, w_1))$$
$$\mathcal{V}(\mathsf{vk}, u_1) = \mathcal{V}_2(\mathsf{vk}, \mathcal{V}_1(\mathsf{vk}, u_1, w_1))$$

Lemma 2.4 (Parallel Composition). Consider relations $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$. For reductions of knowledge $\Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_3 \to \mathcal{R}_4$ we have that $\Pi_1 \times \Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \times \mathcal{R}_3 \to \mathcal{R}_2 \times \mathcal{R}_4$ where

$$\mathcal{P}(\mathsf{pk}, (u_1, u_3), (w_1, w_3)) = (\mathcal{P}_1(\mathsf{pk}, u_1, w_1), \mathcal{P}_2(\mathsf{pk}, u_3, w_3))$$
$$\mathcal{V}(\mathsf{vk}, (u_1, u_3)) = (\mathcal{V}_1(\mathsf{vk}, u_1), \mathcal{V}_2(\mathsf{vk}, u_3))$$

2.5 Refined Reductions of Knowledge

In this section, we formalize an extension to the reductions of knowledge framework: *refined reductions of knowledge*. Refined reductions of knowledge augment structured reductions of knowledge with customizable preconditions on the input instances and postconditions on the output instances. This additional expressivity is needed to capture more complex notions in the reductions of knowledge framework such as incrementally verifiable computation. We prove that refined reductions of knowledge are closed under sequential and parallel composition.

2.5.1 Defining Refined Reductions of Knowledge

Recall that a reduction of knowledge from relation $\mathcal{R}_1$ to relation $\mathcal{R}_2$ is an interactive protocol between a prover and a verifier in which the verifier reduces the task of checking a statement in $\mathcal{R}_1$ to the task of checking a statement in $\mathcal{R}_2$. A *refined reduction of knowledge* is additionally characterized by a binary relation $\sim$, and for input instance u_1 additionally guarantees that the output instance u_2 is such that $u_1 \sim u_2$.

Definition 2.13 (Refined Reduction of Knowledge). Consider (structured) relations $\mathcal{R}_1$ and $\mathcal{R}_2$, over public parameter, structure, instance, and witness tuples and infix binary relation $\sim$. A refined reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ that respects $\sim$ is a (structured) reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ such that for any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$, given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$, $(\mathsf{s}, u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$, $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$, and $(u_2, w_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})$, we have that $u_1 \sim u_2$. We write $\Pi : \mathcal{R}_1 \xrightarrow{\sim} \mathcal{R}_2$ to denote that protocol Π is a reduction of knowledge from relation $\mathcal{R}_1$ to relation $\mathcal{R}_2$ that respects $\sim$.

We additionally define the notion of a *refined relation product*. Refined products augment standard products with an additional binary relation $\sim$ that enforces constraints *between* instances u_1 and u_2.

Definition 2.14 (Refined Relation Product). Consider (structured) relations $\mathcal{R}_1$ and $\mathcal{R}_2$ polynomial-time decidable binary infix relation $\sim$. We define the refined product relation as follows.

$$\mathcal{R}_1 \overset{\sim}{\times} \mathcal{R}_2 = \left\{ (\mathsf{pp}, \mathsf{s}, (u_1, u_2), (w_1, w_2)) \;\middle|\; \begin{array}{l} (\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1, \\ (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2, \\ u_1 \sim u_2 \end{array} \right\}.$$

if $\sim$ does not enforce any additional constraints we simply write $\mathcal{R}_1 \times \mathcal{R}_2$. In this setting we let $\mathcal{R}^\ell$ denote $\mathcal{R} \times \ldots \times \mathcal{R}$ for ℓ times.

2.5.2 Composing Refined Reductions of Knowledge

The goal of this section is to show that *refined* reductions of knowledge are also closed under sequential and parallel composition. This involves additionally reasoning about how the additional refinement relation $\sim$ is preserved under composition.

Lemma 2.5 (Sequential Composition). For refined reductions of knowledge $\Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \xrightarrow{\sim_1} \mathcal{R}_2$ and $\Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_2 \xrightarrow{\sim_2} \mathcal{R}_3$, we have that $\Pi_2 \circ \Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \xrightarrow{\sim_1 \circ \sim_2} \mathcal{R}_3$ where

$$\begin{aligned} \mathcal{P}(\mathsf{pk}, u_1, w_1) &= \mathcal{P}_2(\mathsf{pk}, \mathcal{P}_1(\mathsf{pk}, u_1, w_1)) \\ \mathcal{V}(\mathsf{vk}, u_1) &= \mathcal{V}_2(\mathsf{vk}, \mathcal{V}_1(\mathsf{vk}, u_1, w_1)) \end{aligned}$$

and $u_1 \sim_1 \circ \sim_2 u_3$ if and only if there exists u_2 such that $u_1 \sim_1 u_2$ and $u_2 \sim_2 u_3$.

Proof. By Theorem 2.1, we already have that $\Pi_2 \circ \Pi_1 : \mathcal{R}_1 \to \mathcal{R}_3$. We must additionally show that the relation $\sim_1 \circ \sim_2$ holds. Indeed, consider statement u_1 provided as input to $\Pi_2 \circ \Pi_1$. Let u_2 be the statement produced by Π_1. By the correctness of Π_1, we have that $u_1 \sim_1 u_2$. Let u_3 be the statement produced by Π_2. By the correctness of Π_2, we have that $u_2 \sim_2 u_3$. Therefore, we have that $\Pi_2 \circ \Pi_1 : \mathcal{R}_1 \xrightarrow{\sim_1 \circ \sim_2} \mathcal{R}_2$. □

Lemma 2.6 (Parallel Composition). Consider relations $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$. For reductions of knowledge $\Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \xrightarrow{\sim_1} \mathcal{R}_2$ and $\Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_3 \xrightarrow{\sim_2} \mathcal{R}_4$ we have that $\Pi_1 \times \Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \times \mathcal{R}_3 \xrightarrow{\sim_1 \times \sim_2} \mathcal{R}_2 \times \mathcal{R}_4$ where

$$\begin{aligned} \mathcal{P}(\mathsf{pk}, (u_1, u_3), (w_1, w_3)) &= (\mathcal{P}_1(\mathsf{pk}, u_1, w_1), \mathcal{P}_2(\mathsf{pk}, u_3, w_3)) \\ \mathcal{V}(\mathsf{vk}, (u_1, u_3)) &= (\mathcal{V}_1(\mathsf{vk}, u_1), \mathcal{V}_2(\mathsf{vk}, u_3)) \end{aligned}$$

and $(u_1, u_3) \sim_1 \times \sim_2 (u_2, u_4)$ if and only if $u_1 \sim_1 u_2$ and $u_3 \sim_2 u_4$.

Proof. By Theorem 2.2, we already have that $\Pi_1 \times \Pi_2 : \mathcal{R}_1 \times \mathcal{R}_3 \to \mathcal{R}_2 \times \mathcal{R}_4$. We must additionally show that the relation $\sim_1 \times \sim_2$ holds. Indeed, consider statement (u_1, u_3) provided as input to $\Pi_1 \times \Pi_2$. Let (u_2, u_4) be the statement produced by $\Pi_1 \times \Pi_2$. By the correctness of Π_1, we have that $u_1 \sim_1 u_2$. By the correctness of Π_2, we have that $u_2 \sim_2 u_4$. Therefore, we have that $\Pi_1 \times \Pi_2 : \mathcal{R}_1 \times \mathcal{R}_3 \xrightarrow{\sim_1 \times \sim_2} \mathcal{R}_2 \times \mathcal{R}_4$. □

Chapter 3

The Tensor Reduction of Knowledge

This chapter contains joint work with Bryan Parno [96].

If you really want to impress your friends and confound your enemies, you can invoke tensor products ... People run in terror from the $\otimes$ symbol.

– Brad Osgood,
Lecture Notes for EE 261
The Fourier Transform and its Applications

In this chapter, we develop the tensor reduction of knowledge and applications. In Section 3.1, we provide the necessary background in module theory. In Section 3.2, we formally introduce the tensor reduction, followed by the tensor reduction of knowledge as a generalization of the core reductive step common to most recursive algebraic proofs. In Section 3.3, we instantiate the tensor reduction of knowledge to derive proofs for vector commitments, linear forms, and bilinear forms. In Section 3.4, we show that the linear

algebraic reductions derived from the tensor reduction of knowledge can be composed to derive a proof of knowledge for NP with minimal effort.

3.1 Overview of Module Theory

We start by defining rings and modules. We then define the direct sum and tensor product operations for modules over rings, which is used throughout our development.

Notation (Module Theory). We assume finite, unital, commutative rings and modules with a finite basis throughout. We use $\cong$ to denote that two modules are isomorphic. For ring R and R-modules W and V, let $\hom(W, V)$ denote the R-module of homomorphisms from W to V. For $n \in \mathbb{N}$, we let W^n denote $W \otimes \mathsf{R}^n$ (equivalently $W \oplus \ldots \oplus W$ for n times). We use $\{\delta_i\}$ to denote an orthonormal basis. We refer to elements of modules as tensors. As we use tensors to represent both homomorphisms and objects, for tensors g and a, we use $g(a)$ to denote evaluating the homomorphism tensor g on the object tensor a. For $n \in \mathbb{N}$, let $[n]$ denote $\{1, 2, \ldots, n\}$ and let $[i, n]$ for $i \leq n$ denote $\{i, i+1, \ldots, n\}$. When summing over a variable, we will omit the bounds when clear from context. We write $\langle a, b \rangle$ to denote the inner-product of a and b.

Definition 3.1 (Ring). A ring is a set R together with two binary operations $+$ and $\cdot$ over R that satisfy the following conditions.

(i) $(\mathsf{R}, +)$ is a commutative group.

(ii) Associativity: For all $a, b, c \in \mathsf{R}$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

(iii) Distributivity: For all $a, b, c \in \mathsf{R}$, $(a+b) \cdot c = (a \cdot c) + (b \cdot c)$ and $a \cdot (b+c) = (a \cdot b) + (a \cdot c)$.

The ring is commutative if $a \cdot b = b \cdot a$ for all $a, b \in \mathsf{R}$. The ring is unital if it contains an identity element (denoted 1) such that $1 \cdot a = a \cdot 1 = a$ for all $a \in \mathsf{R}$.

Intuitively, modules are vector spaces over rings. That is, they support a notion of addition, can be scaled by ring elements, and have an identity element. We say a module is an R-module if it is scaled by ring R. Vectors, polynomials, matrices, tensors and scalars all form modules.

Definition 3.2 (Module). Consider commutative ring R. An R-module is a set M together with binary operations $+$ from $M \times M$ to M and $\cdot$ from $\mathsf{R} \times M$ to M that satisfy the following conditions.

(i) $(M, +)$ is a commutative group.

(ii) For all $r, s \in R$ and $m, n \in M$, we have that $(r+s) \cdot m = r \cdot m + s \cdot m$, $(r \cdot s) \cdot m = r \cdot (s \cdot m)$, and $r \cdot (m + n) = r \cdot m + r \cdot n$.

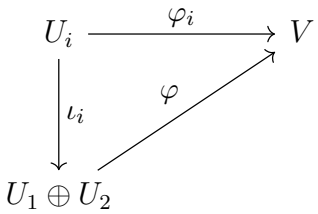
(iii) If R is unital, then $1 \cdot m = m$ for all $m \in M$.

3.1.1 The Direct Sum

Intuitively, a *direct sum* of two R-modules U and V, forms a new R-module denoted $U \oplus V$, which is essentially a Cartesian product of the original modules. Elements of $U \oplus V$ consist of pairs of elements in U and V which are denoted as $u \oplus v$ for $u \in U$ and $v \in V$. For example, for field $\mathbb{F}$, if $U \cong \mathbb{F}^n$ and $V \cong \mathbb{F}^m$ we have that $U \oplus V \cong \mathbb{F}^{n+m}$. We have that $U \oplus V$ forms a module, because we can naturally compute $u_1 \oplus v_1 + u_2 \oplus v_2 = (u_1 + u_2) \oplus (v_1 + v_2)$ and $r \cdot (u \oplus v) = (r \cdot u) \oplus (r \cdot v)$ for $r \in \mathsf{R}$.

Formally, the particular definition of the direct sum depends on the particular modules it is working over. For instance, the direct sum could mean vector concatenation when working over two vector spaces, or mean matrix concatenation when working over two spaces of matrices. Even for a fixed pair of modules, there could exist multiple valid definitions. For instance, for vectors $v_1, v_2 \in \mathbb{F}^n$, we can define $v_1 \oplus v_2$ to be a vector in $\mathbb{F}^{2n}$ or a matrix in $\mathbb{F}^{2 \times n}$. To account for these considerations, we treat the direct sum as an *abstract* operation that can be implemented by any concrete operations that satisfy certain axioms (detailed below). In practice, much like how abstract groups and rings must be instantiated with concrete objects such as elliptic curves and polynomials, the direct sum must be instantiated with a concrete operation that respect the prescribed properties.

Definition 3.3 (Direct Sum). Consider R-modules U_1 and U_2. A direct sum for U_1 and U_2, denoted $\oplus$, is any operation mapping from $U_1 \times U_2$ into a new module, denoted $U_1 \oplus U_2$, such that for natural embedding $\iota_i \in \hom(U_i, U_1 \oplus U_2)$ (where $\iota_1(u_1) \mapsto u_1 \oplus 0$ and $\iota_2(u_2) \mapsto 0 \oplus u_2$), there exists a unique linear map $\varphi \in \hom(U_1 \oplus U_2, V)$ such that for any linear maps $\varphi_i \in \hom(U_i, V)$ the following diagram commutes [1] for $i \in \{1, 2\}$.



Example 3.1 (Direct Sum). Consider field $\mathbb{F}$ and vector spaces $\mathbb{F}^m$ and $\mathbb{F}^n$. Vector concatenation mapping $u_1 \in \mathbb{F}^m$ and $u_2 \in \mathbb{F}^n$ to $(u_1, u_2) \in \mathbb{F}^{m+m}$ is valid direct sum over $\mathbb{F}^m$ and $\mathbb{F}^n$. This is because for any linear maps φ_1 and φ_2, we have that for $\varphi = (\varphi_1, \varphi_2)$

$$\varphi \circ \iota_1(u_1) = (\varphi_1, \varphi_2)(u_1, 0) = \varphi_1(u_1)$$
$$\varphi \circ \iota_2(u_2) = (\varphi_1, \varphi_2)(0, u_2) = \varphi_2(u_2).$$

For the majority of our development, we are interested in taking the direct sum of homomorphisms (represented as tensors). In this situation, we do not need to invoke the abstract definition of this operation, but rather the identities that follow from the axioms.

Lemma 3.1 (Direct Sum of Homomorphisms). Consider commutative ring R. Consider homomorphisms $r \in \hom(U_1, V)$ and $s \in \hom(U_2, V)$ over R-modules. Then $r \oplus s \in$

[1] A diagram is said to commute if all paths along the arrows lead to the same result

$\hom(U_1 \oplus U_2, V)$ is a homomorphism where $(r \oplus s)(u_1 \oplus u_2) = r(u_1) + s(u_2)$. Symmetrically, homomorphisms $r \in \hom(U, V_1)$ and $s \in \hom(U, V_2)$ over R-modules induce a homomorphism $r \oplus s \in \hom(U, V_1 \oplus V_2)$ where $(r \oplus s)(u) = r(u) \oplus s(u)$.

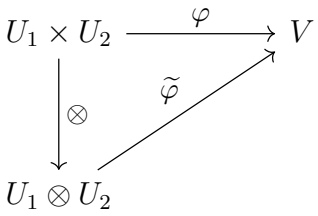
Example 3.2 (Direct Sum of Homomorphisms). Consider group $\mathbb{G}$ of prime order p and corresponding scalar field $\mathbb{F} \cong \mathbb{Z}_p$. We can interpret $\mathbb{G}^n$ as the module of homomorphisms from $\mathbb{F}^n$ to $\mathbb{G}$. In particular, for $g \in \mathbb{G}^n$ we can define $g(a) = \langle g, a \rangle$ for $a \in \mathbb{F}^n$. Then, for $g \in \mathbb{G}^n$ and $h \in \mathbb{G}^m$ we have that $g \oplus h \in \mathbb{G}^n \oplus \mathbb{G}^m \cong \mathbb{G}^{n+m}$ can be interpreted as a map from $\mathbb{F}^{n+m} \cong \mathbb{F}^n \oplus \mathbb{F}^m$ to $\mathbb{G}$. By definition, for $u \in \mathbb{F}^n$ and $v \in \mathbb{F}^m$, we have $(g \oplus h)(u \oplus v) = \langle g \oplus h, u \oplus v \rangle = \langle g, u \rangle + \langle h, v \rangle = g(u) + h(v)$.

3.1.2 The Tensor Product

Intuitively, the tensor product, denoted $\otimes$, can be considered a generalized outer-product that distributes with respect to the direct sum. The tensor product of two modules U and V, forms a new module denoted $U \otimes V$. Elements of $U \otimes V$ include *simple tensors* which are outer products of elements in U and V and are denoted as $u \otimes v$ for $u \in U$ and $v \in V$. The module $U \otimes V$ also contains arbitrary sums of these simple tensors, which are denoted as $\sum_{i \in [\ell]} u_i \otimes v_i$ for $u_1, \ldots, u_\ell \in U$ and $v_1, \ldots, v_\ell \in V$. If $U \cong \mathbb{F}^n$ and $V \cong \mathbb{F}^m$ we have that $U \otimes V \cong \mathbb{F}^{n \times m}$ (i.e., $n \times m$ matrices over $\mathbb{F}$). Simple tensors in $\mathbb{F}^n \otimes \mathbb{F}^m$ consist of outer products of vectors in $\mathbb{F}^n$ and $\mathbb{F}^m$; however, the entire space is generated by sums over such outer products. We have that $U \otimes V$ forms a module because we can naturally add two sums and compute $r \cdot \sum_i u_i \otimes v_i = \sum_i (r \cdot u_i) \otimes v_i = \sum_i u_i \otimes (r \cdot v_i)$.

Formally, as with the direct sum, the particular definition of the tensor product depends on the particular modules is are working over. For instance, the tensor product could mean the outer product when working over vectors and the Kronecker product when working over matrices. To account for this, we also treat the tensor product as *abstract* operations that can be implemented by any concrete operations that satisfy certain axioms (detailed below). In practice, the tensor product must be instantiated with a concrete operation that respects the prescribed properties.

Definition 3.4 (Tensor Product). Consider R-modules U_1 and U_2. A tensor product for U_1 and U_2, denoted $\otimes$, is any operation mapping from $U_1 \times U_2$ into a new module, denoted $U_1 \otimes U_2$, such that for any bilinear map $\varphi : U_1 \times U_2 \to V$ there exists a unique linear map $\widetilde{\varphi} : U_1 \otimes U_2 \to V$ such that the following diagram commutes.



Example 3.3 (Tensor Product). Consider field $\mathbb{F}$ and vector spaces $\mathbb{F}^m$ and $\mathbb{F}^n$. The outer-product mapping $u_1 \in \mathbb{F}^m$ and $u_2 \in \mathbb{F}^n$ to matrix $u_1^\top u_2 \in \mathbb{F}^{m \times n}$ is a valid tensor

product over $\mathbb{F}^m$ and $\mathbb{F}^n$. This is because for any bilinear map from vectors (u_1, u_2), we can derive a corresponding linear map from the matrix $u_1^\top u_2$ that behaves identically.

As with the direct sum, for the majority of our development, we are interested in taking the tensor product of homomorphisms. We recall the identities that follow from the axioms.

Lemma 3.2 (Tensor Product of Homomorphisms). Homomorphisms $r \in \hom(U, X)$ and $s \in \hom(V, Y)$ over R-modules (where R is a commutative ring) induce a homomorphism $r \otimes s \in \hom(U \otimes V, X \otimes Y)$, such that $(r \otimes s)(u \otimes v) = r(u) \otimes s(v)$. By linearity, we have that

$$\Big(\sum_{i\in[I]} r_i \otimes s_i\Big)\Big(\sum_{j\in[J]} u_i \otimes v_i\Big) = \sum_{i\in[I],j\in[J]} r_i(u_j) \otimes s_i(v_j).$$

Example 3.4 (Tensor Product of Homomorphisms). Let $\otimes$ denote the outer product. For prime p and field $\mathbb{F} \cong \mathbb{Z}_p$ we can interpret $\mathbb{F}^n$ as the module of homomorphisms from $\mathbb{F}^n$ to $\mathbb{F}$. In particular, for $f \in \mathbb{F}^n$ we can define $f(a) = \langle f, a\rangle$ for $a \in \mathbb{F}^n$. Then, $f \in \mathbb{F}^n$ and $g \in \mathbb{F}^m$ induce a new map $f \otimes g \in \mathbb{F}^n \otimes \mathbb{F}^m \cong \mathbb{F}^{nm}$ from $\mathbb{F}^{nm}$ to $\mathbb{F} \otimes \mathbb{F} \cong \mathbb{F}$. By definition, for $u \in \mathbb{F}^n$ and $v \in \mathbb{F}^m$, we have $(f \otimes g)(u \otimes v) = (f \cdot g_1 \oplus \ldots \oplus f \cdot g_m)(u \cdot v_1 \oplus \ldots \oplus u \cdot v_m) = \sum_{j\in[m]} f(u) \cdot g_i(v_i) = f(u) \otimes g(v)$.

Lemma 3.3 (Useful Identities). For commutative ring R and R-modules U, V, and W, we have that $(U \otimes V) \otimes W \cong U \otimes (V \otimes W)$, $U \otimes V \cong V \otimes U$, $U \otimes (V \oplus W) \cong (U \otimes V) \oplus (U \otimes W)$, and $\mathsf{R} \otimes U \cong U \otimes \mathsf{R} \cong U$.

3.1.3 Cryptographic Assumptions

We use λ globally to denote the security parameter, and negl to denote negligible functions. For events A and B, we let $\Pr[A] \approx \Pr[B]$ denote that $|\Pr[A] - \Pr[B]| = \mathsf{negl}(\lambda)$. We let PPT denote probabilistic polynomial-time. We write _ to denote unused terms.

For soundness to hold when randomly sampling over rings, the set of admissible values must be constrained. We define a valid sampling set over rings.

Definition 3.5 (Sampling Set [37]). For ring R and R-module M, subset $Q \subseteq \mathsf{R}$ is a sampling set for M if for every $q_1, q_2 \in Q$, the map $\varphi_{q_1,q_2}(m) = (q_1 - q_2) \cdot m$ for $m \in M$ is injective.

For certain relations, to be able to prove knowledge soundness, we will need to rely on computational hardness assumptions. We adapt the bilinear relation assumption [37], which can be viewed as a generalization of the discrete logarithm assumption, and the double pairing assumption [11].

Definition 3.6 (Bilinear Relation Assumption). For ring R, length parameter n, and security parameter λ, consider R-modules U and V such that $|U| = O(2^\lambda)$ and $|V| = O(2^\lambda)$. The bilinear relation assumption holds for (U, V) (w.r.t. tensor product $\otimes$) if given random $u_1, \ldots, u_n \in U$, there exists no polynomial-time algorithm to find non-trivial $v_1, \ldots, v_n \in V$ such that $\sum_{i\in[n]} u_i \otimes v_i = 0$.

Symmetrically, we can consider composite spaces such that given elements from both of the constituent spaces, it is *easy* to check that they satisfy the above relation. This ensures that that the verifier is able to perform its requisite checks efficiently. Throughout our development, we assume the coset equality assumption holds as necessary.

Definition 3.7 (Coset Equality Assumption). For ring R and length parameter n, consider R-modules U and V. The coset equality assumption holds for (U, V) (w.r.t. tensor product $\otimes$) if for any $u_1, \ldots, u_n \in U$ and $v_1, \ldots, v_n \in V$, there exists a polynomial-time algorithm to check $\sum_{i \in [n]} u_i \otimes v_i = 0$.

Example 3.5 (Bilinear Relation Assumption). Suppose U is a group of prime order p and V is the corresponding scalar field $\mathbb{Z}_p$. Let the tensor product between these two modules be defined as scalar multiplication. In this setting, the bilinear relation assumption is equivalent to the discrete logarithm assumption. Alternatively, suppose U and V are prime order groups such that there exists a corresponding pairing operation e from $U \times V$ into some target group. Let the tensor product be defined as this pairing operation. In this setting, the bilinear relation assumption is equivalent to the double pairing assumption.

3.2 The Tensor Reduction of Knowledge

We start by defining a general tensor-based language to capture a large class of linear algebraic statements. We then design a general reduction, the tensor reduction, for such statements, by extending the sum-check protocol [106]. Next, we leverage the tensor reduction to construct the tensor reduction of knowledge, which, for any length vector space of homomorphisms $\hom(W, V)$ and length n, reduces the task of checking knowledge of a preimage of a vector in $\hom(W^n, V)$ to checking knowledge of a preimage in $\hom(W, V)$.

3.2.1 Tensor Evaluation Statements

We observe that proofs of knowledge built around statements over linear algebraic objects — such as matrices, vectors, polynomials, and homomorphisms — typically share hints of symmetry. Our goal is to generalize such statements, and more interestingly generalize interactive reductions for such statements.

Regardless of the underlying linear-algebraic objects, proofs over them tend to only rely on the fact they support some notion of addition and that they can be scaled by elements in a field (and more generally rings). This seems to suggest that designing a reduction over the most general objects that support these operations, namely tensors, would give a single universal protocol for such objects. From an algebraic standpoint, tensors unify objects such as scalars, vectors, matrices, and polynomials. More generally, tensors provide a unifying algebraic object for describing both functions (when viewed as homomorphisms) and objects (when viewed as elements of a module).

Take for instance the vector commitment relation: Given a prime order group $\mathbb{G}$ and an underlying scalar field $\mathbb{F}^n$, a prover claims that for public commitment key $G \in \mathbb{G}^n$ and

commitment $\overline{A}$, it knows a vector $A \in \mathbb{F}^n$ such that $\langle G, A \rangle = \overline{A}$. As the spaces $\mathbb{G}^n$, $\mathbb{F}^n$ and $\mathbb{G}$ are all modules, we can build a corresponding "tensor evaluation" statement

$$G(A) = \overline{A}$$

where G is a tensor in $\mathbb{G}^n$ that maps tensors in $\mathbb{F}^n$ to tensors in $\mathbb{G}$.

Alternatively, suppose in addition to claiming that it knows a vector A underlying a commitment $\overline{A}$ with respect to commitment key G, the prover additionally claims that taking the inner-product of A against some public vector $B \in \mathbb{F}^n$ results in a scalar $\sigma \in \mathbb{F}$. Following our prior reasoning, this can be represented as two tensor evaluation statements: A claim that $G(A) = \overline{A}$ and a claim that $B(A) = \sigma$. But, under the rules of the direct sum (which can be interpreted as a Cartesian product), this is equivalent to applying the tensor $G \oplus B \in \mathbb{G}^n \oplus \mathbb{F}^n$ to A and checking that this results in $\overline{A} \oplus \sigma \in \mathbb{G} \oplus \mathbb{F}$. Namely, we have that the composite statement can be encoded as the following tensor evaluation statement:

$$(G \oplus B)(A) = \overline{A} \oplus \sigma.$$

The flexibility of tensor evaluation statements becomes more salient with the sum-check protocol [106]. In the sum-check protocol, the prover claims for multivariate polynomial $P : \mathbb{F}^n \to \mathbb{F}$ with degree d in each variable that

$$\sum_{x_1,\ldots,x_n \in \{0,1\}} P(x_1, \ldots, x_n) = \sigma \tag{3.1}$$

for some claimed sum $\sigma \in \mathbb{F}$. For $i \in [n]$, consider the tensor $\bigoplus_{j \in [0,d]} x_i^j$ which is just shorthand for the vector $(x_i^0, x_i^1, \ldots, x_i^d)$. Now, consider $\bigotimes_{i \in [n]} \bigoplus_{j \in [0,d]} x_i^j$, which is an n-dimensional matrix populated with all possible products of powers of $x_1, \ldots, x_n$. We can now define a tensor $\boldsymbol{X} = \sum_{x_1,\ldots,x_n \in \{0,1\}} \bigotimes_{i \in [n]} \bigoplus_{j \in [0,d]} x_i^j \in (\mathbb{F}^{d+1})^n$ which encodes all desired evaluation points. Additionally, let $\boldsymbol{P} \in (\mathbb{F}^{d+1})^n$ denote an n-dimensional tensor constituting of the coefficients of P. Specifically, let $\boldsymbol{P}$ contain at index $(j_1, \ldots, j_n)$ the coefficient of P associated with term $x_1^{j_1} x_2^{j_2} \ldots x_n^{j_n}$. Now, we have that checking the original sum-check statement is equivalent to checking the tensor evaluation statement

$$\boldsymbol{P}(\boldsymbol{X}) = \sigma.$$

The three examples above suggest that seemingly disparate linear-algebraic claims can be uniformly viewed as tensor evaluation claims. In light of this, we are interested in designing a reduction for statements of the form $u(w) = v$ for tensors u, w, and v.

3.2.2 The Tensor Reduction

To design a general reduction for tensor statements of the form $u(w) = v$, we start by generalizing the sum-check protocol for tensor evaluation statements. Recall that the sum-check protocol reduces the task of checking the claim in Equation (3.1) to the task of

checking a sum-check claim over a polynomial with one less variable. In particular, the prover begins by sending

$$p(X) = \sum_{x_1,\ldots,x_{n-1}\in\{0,1\}} P(x_1,\ldots,x_{n-1},X)$$

The verifier then checks that $p(0)+p(1)=\sigma$. The verifier must now check that p is consistent with P. To do so, the verifier samples a random $r \leftarrow \mathbb{F}$, and reduces to checking

$$\sum_{x_1,\ldots,x_{n-1}} P(x_1,\ldots,x_{n-1},r) = p(r).$$

In essence, the sum-check protocol leverages the nested structure of polynomials to reduce the task of checking n-variate polynomials to checking $(n-1)$-variate polynomials. This intuition can be more lucidly expressed with the corresponding tensor evaluation statements: the sum-check protocol reduces the task of checking the evaluation of $\boldsymbol{P} \in (\mathbb{F}^{d+1})^n \cong (\mathbb{F}^{d+1})^{n-1} \otimes \mathbb{F}^{d+1}$ (representing P) to the task of checking the evaluation of $\boldsymbol{P}_r \in (\mathbb{F}^{d+1})^{n-1}$ (representing P evaluated on r) and $\boldsymbol{p} \in \mathbb{F}^{d+1}$ (representing p). That is, the sum-check protocol factors the original statement with respect to the tensor product.

The tensor reduction, which we detail below, follows from generalizing the involved spaces to handle arbitrary tensor evaluation statements: for any modules U, U_1, and U_2 such that $U \cong U_1 \otimes U_2$, we derive a mechanism to reduce an evaluation claim in U to an evaluation claim in U_1 and an evaluation claim in U_2. In Appendix 3.5, we show that we can recover the sum-check protocol when instantiating the tensor reduction over multivariate polynomials.

Construction 3.1 (Tensor Reduction). For tensors $u \in \hom(W_1,V_1)\otimes\hom(W_2,V_2)$ of rank I, $w \in W_1 \otimes W_2$ of rank J, and $v \in V_1 \otimes V_2$ over ring R, suppose a verifier would like to check

$$u(w) = v \tag{3.2}$$

where $u = \sum_{i\in[I]} u_{1,i}\otimes u_{2,i}$, and $w = \sum_{j\in[J]} w_{1,j}\otimes w_{2,j}$. By definition, the verifier can check (3.2) by checking $\sum_{i,j} u_{1,i}(w_{1,j})\otimes u_{2,i}(w_{2,j}) = v$. Therefore, the prover begins by computing and sending $v_{1,ij} \leftarrow u_{1,i}(w_{1,j})$ and $v_{2,ij} \leftarrow u_{2,i}(w_{2,j})$ for all $i\in[I], j\in[J]$. The verifier directly checks

$$\sum_{i\in[I],j\in[J]} v_{1,ij}\otimes v_{2,ij} = v.$$

The verifier must still check that $v_{1,ij} = u_{1,i}(w_{1,j})$ and $v_{2,ij} = u_{2,i}(w_{2,j})$ for all i,j. To do so, the verifier takes a random linear combination of these checks by sending random α,β from a valid sampling set $Q \subseteq \mathsf{R}$, and computing $v_1 = \sum_{i,j}\alpha^i\beta^j v_{1,ij}$ and $v_2 = \sum_{i,j}\alpha^i\beta^j v_{2,ij}$. The verifier then outputs (α,β,v_1,v_2), reducing the original check to the task of checking

$$\Big(\sum_i \alpha^i u_{1,i}\Big)\Big(\sum_j \beta^j w_{1,j}\Big) = v_1 \quad \text{and} \quad \Big(\sum_i \alpha^i u_{2,i}\Big)\Big(\sum_j \beta^j w_{2,j}\Big) = v_2.$$

Theorem 3.1 (Tensor Reduction). For tensors $u = \sum_i u_{1,i} \otimes u_{2,i} \in \hom(W_1, V_1) \otimes \hom(W_2, V_2)$ of rank I, $w = \sum_j w_{1,j} \otimes w_{2,j} \in W_1 \otimes W_2$ of rank J, and $v \in V_1 \otimes V_2$ over ring R, the tensor reduction reduces the task of checking

$$u(w) = v$$

to the task of checking

$$\Big(\sum_i \alpha^i u_{1,i}\Big)\Big(\sum_j \beta^j w_{1,j}\Big) = v_1 \quad \text{and} \quad \Big(\sum_i \alpha^i u_{2,i}\Big)\Big(\sum_j \beta^j w_{2,j}\Big) = v_2$$

for verifier output $(\alpha, \beta, v_1, v_2)$. Formally, if the former is true, then the latter is true with probability 1, and if the former is false, then the latter is false with probability at least $1 - \frac{IJ}{|Q|}$. The prover complexity, verifier complexity, and communication complexity are all proportional to IJ.

Proof. This follows from the Schwartz-Zippel Lemma [123] extended to modules [37]. □

At first glance, it may seem that the communication cost of the tensor reduction is *greater* than the size of the witness: the witness only consists of J elements in $W_1 \otimes W_2$, but the prover sends IJ elements in V_1 and V_2. This is reconciled by the fact that elements of V_1 and V_2 are intended to be significantly smaller than elements in $W_1 \otimes W_2$. For instance, elements in $W_1 \otimes W_2$ may be long vectors that are mapped to short commitments in V_1 and V_2.

To build intuition for where tensor reductions are useful, we explain how to instantiate the tensor reduction to reconstruct the vector commitment reduction of knowledge presented in Section 1.4.

Example 3.6 (Vector Commitment Reduction of Knowledge). We construct a reduction of knowledge from $\mathcal{R}_{\mathsf{VC}}(n)$ to $\mathcal{R}_{\mathsf{VC}}(n/2)$ for $n = 2^i$ where $i \geq 1$. Consider group $\mathbb{G}$ of prime order p, and corresponding scalar field $\mathbb{F} \cong \mathbb{Z}_p$. Consider some public key $G \in \mathbb{G}^n$. Suppose a verifier would like to check for some commitment $\overline{A} \in \mathbb{G}$, that the prover knows vector $A \in \mathbb{F}^n$ such that $G(A) = \overline{A}$ where $G(A)$ is defined to be $\langle G, A \rangle$.

We observe that $\mathbb{G}^n \cong \mathbb{G}^{n/2} \otimes \mathbb{F}^2$ and $\mathbb{F}^n \cong \mathbb{F}^{n/2} \otimes \mathbb{F}^2$. Let $\{\delta_1, \delta_2\}$ be an orthonormal basis for $\mathbb{F}^2$ (i.e., we have that $\delta_i(\delta_j) = 1$ when $i = j$ and 0 otherwise). Then, we have that $G = G_1 \otimes \delta_1 + G_2 \otimes \delta_2$ and $A = A_1 \otimes \delta_1 + A_2 \otimes \delta_2$ for some $G_1, G_2 \in \mathbb{G}^{n/2}$ and $A_1, A_2 \in \mathbb{F}^{n/2}$. These terms can be interpreted as the first and second half of vectors G and A. Therefore, the verifier can equivalently check

$$\Big(\sum_i G_i \otimes \delta_i\Big)\Big(\sum_j A_j \otimes \delta_j\Big) = \overline{A}.$$

Applying the tensor reduction with respect to this decomposition, we have that the prover sends to the verifier $G_i(A_j), \delta_i(\delta_j)$ for $i, j \in \{1, 2\}$. Explicitly, letting $\overline{A}_{ij} = G_i(A_j)$, the prover sends the terms $(\overline{A}_{11}, 1)$, $(\overline{A}_{12}, 0)$, $(\overline{A}_{21}, 0)$, and $(\overline{A}_{22}, 1)$. We recognize that the first and last terms correspond with the first and second half of commitment $\overline{A}$, and the middle two terms are cross terms.

Upon receiving these terms, the verifier checks that

$$\overline{A}_{11} \otimes 1 + \overline{A}_{12} \otimes 0 + \overline{A}_{21} \otimes 0 + \overline{A}_{22} \otimes 1 = \overline{A}.$$

The verifier then samples and sends random $\alpha, \beta \leftarrow \mathbb{F}$, and sets the new statements to be checked to be $(G_1 + \alpha G_2)(A_1 + \beta A_2) = \sum_{i,j \in \{1,2\}} \overline{A}_{ij} \cdot \alpha^i \beta^j$ and $(\delta_1 + \alpha\delta_2)(\delta_1 + \beta\delta_2) = 1 + (\beta + \alpha) \cdot 0 + \alpha\beta \cdot 1$. The latter check holds immediately. As for the former check, the prover and verifier compute and output the new statement $G' \leftarrow G_1 + \alpha \cdot G_2 \in \mathbb{G}^{n/2}$ and $\overline{A} \leftarrow \sum_{i,j \in \{1,2\}} \overline{A}_{ij} \cdot \alpha^i \beta^j$. The prover privately computes and outputs the new witness vector $A' \leftarrow A_1 + \beta A_2 \in \mathbb{F}^{n/2}$. Now, it is sufficient for the verifier to check that the prover knows $A' \in \mathbb{F}^{n/2}$ such that

$$G'(A') = \overline{A}'.$$

3.2.3 The Tensor Reduction of Knowledge

By generalizing Example 3.6 for arbitrary tensor statements, we arrive at the tensor reduction of knowledge, which is unconditionally secure. We start by defining the tensor relation which fixes the homomorphism and image as a statement and the preimage as the witness.[2] We then construct the tensor reduction of knowledge, which for a vector space of homomorphisms U and length n, reduces the task of checking knowledge of a preimage of a homomorphism in U^n to the task of checking knowledge of a preimage of a homomorphism in U. In the upcoming section, we show that the tensor reduction of knowledge can be instantiated to derive reductions of knowledge for various linear algebraic statements.

Definition 3.8 (Tensor Relation). For ring R and R-modules U, W and V, such that $U \cong \hom(W, V)$ we define the tensor relation for U as follows

$$\mathcal{R}(U) = \left\{ \; ((u, v), w) \;\middle|\; \begin{array}{l} u \in U, v \in V, w \in W, \\ u(w) = v \end{array} \right\}$$

Construction 3.2 (Tensor Reduction of Knowledge). Consider field $\mathbb{F}$, length parameter n, and $\mathbb{F}$-modules W and V. We construct a reduction of knowledge from $\mathcal{R}(\hom(W^n, V))$ to $\mathcal{R}(\hom(W, V))$. Let $\{\delta_i\}$ be an orthonormal basis for $\mathbb{F}^n$. Suppose the prover and verifier are provided statement $u = \sum_i u_i \otimes \delta_i \in \hom(W^n, V)$, and $v \in V$. Additionally, suppose the prover is provided an alleged witness $w = \sum_j w_j \otimes \delta_j \in W^n$ such that

$$((u, v), w) \in \mathcal{R}(\hom(W^n, V)).$$

The prover and verifier run a single tensor reduction on the equivalent statement

$$\Big(\sum_{i \in [n]} u_i \otimes \delta_i\Big)\Big(\sum_{j \in [n]} w_j \otimes \delta_j\Big) = v.$$

[2]The tensor relation can be formally understood as a ternary relation where any public parameters are ignored. This makes it compatible with the reductions of knowledge framework which works over ternary relations defined over public parameter, statement, and witness tuples.

At the end of tensor reduction, the verifier outputs $(\alpha, \beta, v', \_)$. The prover and verifier compute $u' = \sum_i \alpha^i \cdot u_i$ and set the output statement to be (u', v'). The prover additionally computes the output witness $w' = \sum_j \beta^j \cdot w_j$ as dictated by the tensor reduction. Now, to check the original statement, it is sufficient for the verifier to check that the prover knows w' such that

$$((u', v'), w') \in \mathcal{R}(\hom(W, V)).$$

Theorem 3.2 (Tensor Reduction of Knowledge). For field $\mathbb{F}$, length parameter n, and $\mathbb{F}$-modules W and V, Construction 3.2 is a reduction of knowledge from $\mathcal{R}(\hom(W^n, V))$ to $\mathcal{R}(\hom(W, V))$.

Proof Intuition. Consider instance $u = \sum_{i\in[n]} u_i \otimes \delta_i$ and v. We prove knowledge soundness via tree extraction (Lemma 2.2). That is, we construct extractor χ that outputs w such that $u(w) = v$ given a tree of accepting transcripts and corresponding output prover witnesses.

Suppose the extractor χ is provided with n^2 accepting transcripts τ_{mk} with the same prover's first message $\{(v_{1,ij}, v_{2,ij}) | i, j \in [n]\}$ and with randomness (α_m, β_{mk}) for $m \in [n], k \in [n]$. Let $w'_{mk} \in W'$ for $k \in [n^2]$ denote the corresponding satisfying witnesses. For $m \in [n]$, the extractor solves for $w_{mj} \in W$ for $j \in [n]$ such that $\sum_{j\in[n]} \beta^j_{mk} w_{mj} = w'_{mk}$ for $k \in [n]$ using an inverse Vandermonde matrix (where invertibility is afforded by working over a field). The extractor then computes a_{mj} for all $m \in [n], j \in [n]$ such that for all $i \in [n]$, $\sum_{m\in[n]} \alpha^i_m a_{mj} = v_{2,ij}$. Next, the extractor computes

$$w \leftarrow \sum_{l\in[n]} \sum_{m\in[n]} \sum_{j\in[n]} a_{mj} \cdot \alpha^l_m \cdot w_{mj} \otimes \delta_l.$$

By textbook algebra, we can show that w is indeed a satisfying witness. □

Proof. Consider instance $u = \sum_{i\in[n]} u_i \otimes \delta_i$ and v. We prove knowledge soundness via tree extraction (Lemma 2.2). That is, we construct extractor χ that outputs w such that $u(w) \cong v$ given a tree of accepting transcripts and corresponding output prover witnesses.

Suppose the extractor χ is provided with n^2 accepting transcripts τ_{mk} with the same prover's first message

$$\{(v_{1,ij}, v_{2,ij}) | i \in [n], j \in [n]\}$$

and with randomness (α_m, β_{mk}) for $m, k \in [n]$. Let $w'_{mk} \in W$ for $m, k \in [n]$ denote the corresponding satisfying witnesses. For $m \in [n]$, the extractor solves for $w_{mj} \in W$ for $j \in [n]$ such that

$$\sum_{j\in[n]} \beta^j_{mk} w_{mj} = w'_{mk} \tag{3.3}$$

for $k \in [n]$ using an inverse Vandermonde matrix (where invertibility is afforded by working over a field). Because w'_{mk} is a satisfying witness, by construction of the tensor reduction, for all $m, k \in [n]$ we have that

$$\Big(\sum_i \alpha^i_m u_i\Big)\Big(w'_{mk}\Big) = \Big(\sum_{i,j} \alpha^i_m \beta^j_{mk} v_{1,ij}\Big).$$

Then, by Equation (3.3) we have for all $m, k \in [n]$

$$\Big(\sum_i \alpha_m^i u_i\Big)\Big(\sum_j \beta_{mk}^j w_{mj}\Big) = \Big(\sum_{i,j} \alpha_m^i \beta_{mk}^j v_{1,ij}\Big).$$

Rearranging terms, we have that

$$\sum_j \Big(\sum_i \alpha_m^i u_i(w_{mj})\Big) \cdot \beta_{mk}^j = \sum_j \Big(\sum_i \alpha_m^i v_{1,ij}\Big) \cdot \beta_{mk}^j. \tag{3.4}$$

Thus, for each $m \in [n]$, we can treat both sides of Equation 3.4 as polynomials evaluated over $\{\beta_{mk} | k \in [n]\}$. Because equality holds for $k \in [n]$ distinct evaluations, we have that for all $m, j \in [n]$

$$\Big(\sum_i \alpha_m^i u_i\Big)\Big(w_{mj}\Big) = \Big(\sum_i \alpha_m^i v_{1,ij}\Big). \tag{3.5}$$

To compute a satisfying witness, the extractor first computes a_{mj} for all $m, j \in [n]$ such that for all $i \in [n]$

$$\sum_{m \in [n]} \alpha_m^i a_{mj} = v_{2,ij}. \tag{3.6}$$

Next, the extractor computes

$$w \leftarrow \sum_{l \in [n]} \sum_{m \in [n]} \sum_{j \in [n]} a_{mj} \cdot \alpha_m^l \cdot w_{mj} \otimes \delta_l. \tag{3.7}$$

We must now show that w is a satisfying witness. Indeed, we have

$$\begin{aligned}
u(w) &= \Big(\sum_i u_i \otimes \delta_i\Big)\Big(\sum_{l,m,j} a_{mj} \cdot \alpha_m^l \cdot w_{mj} \otimes \delta_l\Big) && \text{By (3.7).} \\
&= \sum_{i,l,m,j} a_{mj} \cdot \alpha_m^l \cdot u_i(w_{mj}) \otimes \delta_i(\delta_l) \\
&= \sum_{i,m,j} a_{mj} \cdot \alpha_m^i \cdot u_i(w_{mj}) && \text{By } \delta_i(\delta_l) = 0 \text{ for } i \neq l. \\
&= \sum_{m,j} a_{mj} \cdot \sum_i \alpha_m^i \cdot u_i(w_{mj}) \\
&= \sum_{m,j} a_{mj} \cdot \sum_i \alpha_m^i \cdot v_{1,ij} && \text{By (3.5).} \\
&= \sum_{i,j} v_{1,ij} \cdot \sum_m \alpha_m^i a_{mj} \\
&= \sum_{i,j} v_{1,ij} \cdot v_{2,ij} && \text{By (3.6).} \\
&= v. && \text{By the verifier's check.}
\end{aligned}$$

□

3.3 Instantiating the Tensor Reduction of Knowledge

In this section, we demonstrate a unifying view of existing recursive algebraic proofs by deriving them by instantiating the tensor reduction of knowledge over the appropriate structures. We additionally derive new reductions of knowledge for bilinear forms by extending our techniques. We additionally discuss concrete modules each of these reductions can be instantiated over. In Section 3.4, we show how to stitch together these reductions to derive a proof of knowledge for NP.

3.3.1 Vector Commitments and Linear Forms

We start by generalizing the vector commitment relation from Section 1.4 and then discuss how to succinctly derive the vector commitment reduction of knowledge via the tensor reduction of knowledge. We then adapt the vector commitment reduction for linear forms. The high level approach is to first split all checks over size n vectors into k checks over size n/k vectors. These checks are then folded using a random linear combination. How exactly the vectors are split and folded is abstracted away by the tensor reduction of knowledge.

Consider size parameter $n \in \mathbb{N}$, and consider $\mathbb{F}$-modules $\mathbb{G}$ and $\mathbb{H}$ for field $\mathbb{F}$ such that $\mathbb{G} \cong \hom(\mathbb{H}, \mathbb{G} \otimes \mathbb{H})$. For public key $G \in \mathbb{G}^n$, and commitment $\overline{H} \in \mathbb{G} \otimes \mathbb{H}$, suppose a verifier would like to check that a prover knows $H \in \mathbb{H}^n$ such that $\sum_i G_i \otimes H_i = \overline{H}$. For example, suppose $\mathbb{G}$ is a group of prime order p where the discrete logarithm is hard, $\mathbb{H}$ and $\mathbb{F}$ are $\mathbb{Z}_p$, and $\otimes$ represents scalar multiplication. Then, this amounts to checking knowledge of the opening for a Pedersen commitment. Recall that the prover's claim can be expressed as a tensor statement $G(H) = \overline{H}$. Therefore, because $G \in \mathbb{G}^n$, we define the generalized vector commitment relation as the tensor relation over homomorphisms in $\mathbb{G}^n$.

Definition 3.9 (Generalized Vector Commitment Relation). For length $n \in \mathbb{N}$ and group $\mathbb{G}$, the vector commitment relation is defined to be $\mathcal{R}(\mathbb{G}^n)$.

Construction 3.3 (Vector Commitment Reduction of Knowledge). Because $\mathbb{G}^n \cong (\mathbb{G}^{n/k})^k$, we can directly apply the tensor reduction of knowledge to get a reduction from $\mathcal{R}(\mathbb{G}^n)$ to $\mathcal{R}(\mathbb{G}^{n/k})$.

Suppose that in addition to checking that the prover knows a vector opening to a commitment, the verifier would like to additionally check some public linear combination of the prover's opening. In particular, for public vector $A \in \mathbb{F}^n$, and $\sigma \in \mathbb{H}$, suppose the verifier would like to additionally check that $A(H) = \sigma$ where $A(H)$ is defined to be $\sum_{i \in [n]} A_i \otimes H_i$. For example, if $\otimes$ represents scalar multiplication, then this amounts to checking an inner-product. Recall, from Section 3.2, that this is equivalent to checking $(G \oplus A)(H) = \overline{H} \oplus \sigma$. Because $G \oplus A \in \mathbb{G}^n \oplus \mathbb{F}^n$, we define the linear forms relation as follows.

Definition 3.10 (Linear Forms Relation). For length n and $\mathbb{F}$-module $\mathbb{G}$ for field $\mathbb{F}$, let $\mathsf{LF}_n = \mathbb{G}^n \oplus \mathbb{F}^n$. The linear forms relation is defined to be $\mathcal{R}(\mathsf{LF}_n)$.

Construction 3.4 (Linear Forms Reduction of Knowledge). Consider $n, k \in \mathbb{N}$ such that k divides n. We construct a reduction of knowledge from $\mathcal{R}(\mathsf{LF}_n)$ to $\mathcal{R}(\mathsf{LF}_{n/k})$. In particular, we have that

$$\mathsf{LF}_n = (\mathbb{G} \oplus \mathbb{F})^n \cong (\mathbb{G} \oplus \mathbb{F})^{(n/k)\cdot k} = (\mathsf{LF}_{n/k})^k.$$

Therefore, the prover and verifier can apply the tensor reduction of knowledge with respect to this decomposition to reduce the task of checking a statement in $\mathcal{R}(\mathsf{LF}_n)$ to the task of checking a statement in $\mathcal{R}(\mathsf{LF}_{n/k})$.

Lemma 3.4 (Linear Forms Reduction of Knowledge). Construction 3.4 is a reduction of knowledge from LF_n to $\mathsf{LF}_{n/k}$ with $O(n)$ prover and verifier time complexity and $O(k^2)$ communication complexity.

As discussed in Section 1.4, we can construct a base case proof of knowledge for LF_1 where the prover directly reveals the witness. Thus, we have the following.

Corollary 3.1 (Linear Forms Proof of Knowledge). Consider $n, k \in \mathbb{N}$ such that k divides n. Let Π_{LF} be a reduction of knowledge from $\mathcal{R}(\mathsf{LF}_n)$ to $\mathcal{R}(\mathsf{LF}_{n/k})$. Let Π_{base} be a proof of knowledge for $\mathcal{R}(\mathsf{LF}_1)$. Then

$$\Pi_{\mathsf{base}} \circ \underbrace{\Pi_{\mathsf{LF}} \circ \ldots \circ \Pi_{\mathsf{LF}}}_{\log_k n \text{ times}}$$

is a proof of knowledge for LF_n with $O(n)$ prover and verifier time complexity and $O(k^2 \cdot \log_k n)$ communication complexity.

3.3.2 Bilinear Forms

We extend the above methodology to develop a new reduction for bilinear forms. Recall that the public parameters consist of public key $G \in \mathbb{G}^m$, and the statement consists of matrix $M \in \mathbb{F}^{m \times m}$, commitments $\overline{A}, \overline{B} \in \mathbb{G}$, and scalar $\sigma \in \mathbb{F}$. A witness $(A, B) \in \mathbb{F}^m$ is satisfying if $A^\top M B = \sigma$, $\langle G, A \rangle = \overline{A}$, and $\langle G, B \rangle = \overline{B}$.

Below, we define a slight generalization where the length n of the vector B is some fraction of the length m. The key G is first (partially) compressed with respect to some public random vector $r \in \mathbb{F}^{m/n}$ to produce a new key $H \in \mathbb{G}^n$. This key is instead used to commit to the vector B. Our bilinear forms reduction will recursively compress G and B until $n = 1$. At this point the bilinear forms statement can be reduced to a linear forms statement.

Definition 3.11 (Bilinear Forms, Original). Consider $\mathbb{F}$-module $\mathbb{G}$ for field $\mathbb{F}$. We define the bilinear forms relation, $\mathcal{R}_{\mathsf{Bil}}$, characterized by m rows and n columns as follows.

$$\mathcal{R}_{\mathsf{Bil}(m,n)} = \left\{ (G, (M, r, \overline{A}, \overline{B}, \sigma), (A, B)) \;\middle|\; \begin{array}{l} G \in \mathbb{G}^m, M \in \mathbb{F}^{m \times n}, r \in \mathbb{F}^{m/n}, \\ (\overline{A}, \overline{B}) \in \mathbb{G}, \sigma \in \mathbb{F}, \\ A^\top M B = \sigma, G(A) = \overline{A}, G(r \otimes B) = \overline{B} \end{array} \right\}$$

Unlike vector commitments and linear forms, the bilinear forms relation cannot be encoded directly as a tensor evaluation statement. Our approach is to encode the original statement as the *related* statement,

$$(G \otimes H \oplus \boldsymbol{M})(A \otimes B) = (\overline{A} \otimes \overline{B} \oplus \sigma), \tag{3.8}$$

where $\boldsymbol{M} \in \mathbb{F}^m \otimes \mathbb{F}^n$ is a tensor such that $\boldsymbol{M}(A \otimes B) = A^\top M B$ and $H = G(r) \in \mathbb{G}^n$. The tensor-based statement implies checking the original statement so long as we additionally stipulate that the bilinear relation assumption holds for $(\mathbb{G}, \mathbb{F})$, and $(\mathbb{G}, \mathbb{G})$. Then, we can utilize the tensor reduction of knowledge to reduce the corresponding tensor relation $\mathcal{R}(\mathbb{G}^m \otimes \mathbb{G}^n \oplus \mathbb{F}^m \otimes \mathbb{F}^n)$.

In practice, $\mathbb{G}$ can be a symmetric bilinear group with the pairing operation acting as the tensor product and $\mathbb{G} \otimes \mathbb{G}$ denoting the target group. In this setting, the bilinear relation assumptions are equivalent to the discrete logarithm assumption over $\mathbb{G}$ and the double pairing assumption [11] over $(\mathbb{G}, \mathbb{G})$.

The computational hardness assumptions are a critical detail for arguing that checking Equation (3.8) is sufficient to check the original relation: the unconditional knowledge soundness property of the tensor reduction of knowledge only guarantees that the prover knows *some* satisfying witness in $\mathbb{F}^m \otimes \mathbb{F}^n$ which may be of the form $\sum_i A_i \otimes B_i$ (i.e., not a simple tensor). While this is a valid witness for the corresponding tensor statement, it is *not* a valid witness for the original statement. However, by assuming that the commitment scheme is computationally binding, we can argue that all A_i values must be the same. Leveraging this, we can show that the prover must know a single A and B vector that satisfies the statement. Formally, we define the bilinear forms relation as follows.

Definition 3.12 (Bilinear Forms, Tensor). Consider $n, m \in \mathbb{N}$, and consider $\mathbb{F}$-module $\mathbb{G}$ for field $\mathbb{F}$ such that the bilinear relation assumption holds for $(\mathbb{G}, \mathbb{F})$, and $(\mathbb{G}, \mathbb{G})$. Let $\mathsf{BF}_{m,n} = (\mathbb{G}^m \otimes \mathbb{G}^n) \oplus (\mathbb{F}^m \otimes \mathbb{F}^n)$. We define the (tensor-based) bilinear form relation as the corresponding tensor relation $\mathcal{R}(\mathsf{BF}_{m,n})$.

Next, we show how to recursively reduce $\mathcal{R}_{\mathsf{Bil}(m,n)}$ to $\mathcal{R}(\mathsf{LF}_m)$. To do so, we construct a reduction from $\mathcal{R}_{\mathsf{Bil}(m,n)}$ to $\mathcal{R}_{\mathsf{Bil}(m,n/k)}$, which internally uses the tensor reduction of knowledge from $\mathcal{R}(\mathsf{BF}_{m,n})$ to $\mathcal{R}(\mathsf{BF}_{m,n/k})$. We then construct a base case reduction from $\mathcal{R}_{\mathsf{Bil}(m,1)}$ to $\mathcal{R}(\mathsf{LF}_m)$.

Construction 3.5 (Bilinear Forms Reduction of Knowledge). Consider $n, k \in \mathbb{N}$ such that k divides n. We reduce from $\mathcal{R}_{\mathsf{Bil}(m,n)}$ to $\mathcal{R}_{\mathsf{Bil}(m,n/k)}$.

The generator samples public key $G \leftarrow \mathbb{G}^m$. Suppose that the prover and verifier take as input statement $(M, r, \overline{A}, \overline{B}, \sigma)$ and the prover additionally takes as input and witness (A, B) such that

$$(G, (M, r, \overline{A}, \overline{B}, \sigma), (A, B)) \in \mathcal{R}_{\mathsf{Bil}(m,n)}$$

The prover and verifier begin by encoding the statement and witness as

$$((G \otimes H \oplus \boldsymbol{M}, \overline{A} \otimes \overline{B} \oplus \sigma), A \otimes B) \in \mathcal{R}(\mathsf{BF}_{m,n})$$

where $\boldsymbol{M} \in \mathbb{F}^m \otimes \mathbb{F}^n$ is such that $\boldsymbol{M}(A \otimes B) = A^\top M B$ and $H = G(r) \in \mathbb{G}^n$.

We observe that

$$\mathsf{BF}_{m,n} = \mathbb{G}^m \otimes \mathbb{G}^n \oplus \mathbb{F}^m \otimes \mathbb{F}^n \cong (\mathbb{G}^m \otimes \mathbb{G}^{n/k} \oplus \mathbb{F}^m \otimes \mathbb{F}^{n/k})^k = (\mathsf{BF}_{m,n/k})^k.$$

Therefore, the prover and verifier can apply the tensor reduction of knowledge with respect to this decomposition and reduce to the task of checking a statement in $\mathcal{R}(\mathsf{BF}_{m,n/k})$. At a high level, the tensor reduction prover and verifier partition $\boldsymbol{M}$ and H into k sets of columns and the prover partitions B into k corresponding sets of rows. The prover and verifier then take a random linear combination of these sets against weights $(s, s^2, \ldots, s^k)$ for some randomness $s \in \mathbb{F}$. By linearity, we have that the output statement is of the form

$$((G \otimes H' \oplus \boldsymbol{M}', \overline{A} \otimes \overline{B}' \oplus \sigma'), A \otimes B') \in \mathcal{R}(\mathsf{BF}_{m,n/k})$$

for some $H' = H((s, \ldots, s^k)) \in \mathbb{G}^{n/k}$, $\boldsymbol{M}' = \boldsymbol{M}((s, \ldots, s^k)) \in \mathbb{F}^m \otimes \mathbb{F}^{n/k}$, $\overline{B}' \in \mathbb{G}$, $\sigma' \in \mathbb{F}$, and $B' \in \mathbb{F}^{n/k}$. Together, the prover and verifier output the decoded statement $(M', (r \otimes (s, \ldots, s^k)), \overline{A}, \overline{B}', \sigma')$ and witness (A, B'). Now it is sufficient for the verifier to check that the prover knows (A, B') such that.

$$(G, (M', (r \otimes (s, \ldots, s^k)), \overline{A}, \overline{B}', \sigma'), (A, B')) \in \mathcal{R}_{\mathsf{Bil}(m,n/k)}.$$

Lemma 3.5 (Bilinear Forms Reduction of Knowledge). Construction 3.5 is a reduction of knowledge from $\mathcal{R}_{\mathsf{Bil}(m,n)}$ to $\mathcal{R}_{\mathsf{Bil}(m,n/k)}$ with $O(mn)$ prover and verifier time complexity and $O(k^2)$ communication complexity.

Proof Intuition. Completeness, prover and verifier time complexity, and communication complexity follow from the corresponding properties of the tensor reduction of knowledge. By the composability of reductions (Theorem 2.1), the extractor can retrieve a satisfying witness $\sum_i A'_i \otimes B'_i \in \mathbb{F}^n \otimes \mathbb{F}^m$ for the underlying tensor reduction of knowledge. By the bilinear relation assumption over $(\mathbb{G}, \mathbb{F})$ and $(\mathbb{G}, \mathbb{G})$, the witness must be of the form $A \otimes B$ for some efficiently computable $A \in \mathbb{F}^m$ and $B \in \mathbb{F}^n$. □

Proof. Completeness, prover time complexity, verifier time complexity and communication complexity follow by the corresponding properties of the tensor reduction of knowledge.

As for knowledge soundness, consider expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$. Suppose that $G \leftarrow \mathcal{G}(\lambda)$ and $((M, r, \overline{A}, \overline{B}, \sigma), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$. Suppose additionally that

$$\Pr[(G, \langle \mathcal{P}^*, \mathcal{V} \rangle (G, (M, r, \overline{A}, \overline{B}, \sigma), \mathsf{st})) \in \mathcal{R}_{\mathsf{Bil}(m,n/k)}] = \epsilon$$

We must construct an extractor $\mathcal{E}$ that succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$. Formally, we must have

$$\Pr[(G, \mathcal{E}(G, (M, r, \overline{A}, \overline{B}, \sigma), \mathsf{st})) \in \mathcal{R}_{\mathsf{Bil}(m,n)}] = \epsilon - \mathsf{negl}(\lambda).$$

By translating statements between $\mathcal{R}_{\mathsf{Bil}}$ and $\mathcal{R}(\mathsf{BF})$ as described in Construction 3.5, Prover $\mathcal{P}^*$ implies a malicious prover $\mathcal{P}^{**}$ that succeeds with probability ϵ for the underlying tensor reduction of knowledge. Thus, $\mathcal{E}$ runs the corresponding extractor for the tensor reduction of knowledge which succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$. By construction, the

tensor reduction extractor is provided a tree of accepting transcripts. Any branch, after appropriate translation of statements, can be parsed to retrieve $A, B' \in \mathbb{F}^m, \mathbb{F}^{n/k}$ such that

$$(G, (M', r', \overline{A}, \overline{B}', \sigma'), (A, B')) \in \mathcal{R}_{\mathsf{Bil}(m,n/k)}.$$

for some $M' \in \mathbb{F}^m \otimes \mathbb{F}^{n/k}$, $r' \in \mathbb{F}^{k \cdot m/n}$, $\overline{B}' \in \mathbb{G}$, and $\sigma' \in \mathbb{F}$. By the bilinear relation assumption over $(\mathbb{G}, \mathbb{F})$ the vector A in any given branch is the same with probability $1 - \mathsf{negl}(\lambda)$. Let $A = (a_1, \ldots, a_n)$.

With probability $\epsilon - \mathsf{negl}(\lambda)$, the tensor reduction extractor succeeds in producing $\sum_i A'_i \otimes B'_i \in \mathbb{F}^n \otimes \mathbb{F}^m$ such that

$$\Big(G \otimes H \oplus \boldsymbol{M}\Big)\Big(\sum_i A'_i \otimes B'_i\Big) = \overline{A} \otimes \overline{B} \oplus \sigma$$

for $H = G(r)$ with probability $\epsilon - \mathsf{negl}(\lambda)$. We will show that due to the bilinear relation assumption over $(\mathbb{G}, \mathbb{F})$ and $(\mathbb{G}, \mathbb{G})$, the witness must be of the form $A \otimes B$ for some efficiently computable B. Indeed, rearranging we have that

$$\sum_i A'_i \otimes B'_i = \sum_i \delta_i \otimes B_i$$

for canonical basis $\{\delta_i\}$ for $\mathbb{F}^n$ and some $B_i \in \mathbb{F}^m$. Then, we have

$$\Big(G \otimes H\Big)\Big(\sum_i \delta_i \otimes B_i\Big) = \sum_i G_i \otimes \overline{B}_i$$

where $\overline{B}_i = H(B_i)$. Additionally, we have

$$\overline{A} \otimes \overline{B} = \Big(\sum_i a_i \cdot G_i\Big) \otimes \overline{B} = \sum_i G_i \otimes (a_i \cdot \overline{B}).$$

By the bilinear relation assumption over $(\mathbb{G}, \mathbb{G})$ we have $\overline{B}_i = a_i \cdot \overline{B}$ for all $i \in [n]$ with overwhelming probability. This in turn implies $H(a_i^{-1} \cdot B_i) = \overline{B}$ for all $i \in [n]$. Then, by the bilinear relation assumption over $(\mathbb{G}, \mathbb{F})$, we have that

$$a_1^{-1} \cdot B_1 = \ldots = a_n^{-1} \cdot B_n$$

with overwhelming probability. Let $B = a_i^{-1} \cdot B_i$ denote the above value. Then we have that $A \otimes B$ is a satisfying witness because

$$A \otimes B = \sum_i a_i \cdot \delta_i \otimes B = \sum_i \delta_i \otimes B_i = \sum_i A'_i \otimes B'_i.$$

□

Construction 3.6 (Bilinear Forms Base Case). We construct a reduction of knowledge from $\mathcal{R}_{\mathsf{Bil}(m,1)}$ to $\mathcal{R}(\mathsf{LF}_m)$. Once again the generator samples public key $G \leftarrow \mathbb{G}^m$. Consider statement $(M, r, \overline{A}, \overline{B}, \sigma)$ and alleged witness (A, B). The prover begins the reduction by directly sending B to the verifier. The verifier immediately checks that $H(B) = \overline{B}$ for $H = G(r)$. Additionally, as $M \in \mathbb{F}^m$ and $B \in \mathbb{F}$, the verifier computes the vector $V \leftarrow M \cdot B$. The verifier is left with checking that the prover knows $A \in \mathbb{F}^m$ such that $G(A) = \overline{A}$ and $V(A) = \sigma$. This is equivalent to checking that $((G \oplus V, \overline{A} \oplus \sigma), A) \in \mathcal{R}(\mathsf{LF}_m)$.

Lemma 3.6 (Bilinear Forms Base Case). Construction 3.6 is a reduction of knowledge from $\mathcal{R}_{\mathsf{Bil}(m,1)}$ to $\mathcal{R}(\mathsf{LF}_m)$ with $O(m)$ prover and verifier time complexity and $O(1)$ communication complexity.

Corollary 3.2 (Bilinear Forms to Linear Forms). Consider $n, k \in \mathbb{N}$ such that k divides n. Let Π_{Bil} be the reduction of knowledge from $\mathcal{R}_{\mathsf{Bil}(m,n)}$ to $\mathcal{R}_{\mathsf{Bil}(m,n/k)}$. Let Π_{base} be the reduction of knowledge from $\mathcal{R}(\mathsf{Bil}(m,1))$ to $\mathcal{R}(\mathsf{LF}_m)$. Then

$$\Pi_{\mathsf{base}} \circ \underbrace{\Pi_{\mathsf{Bil}} \circ \ldots \circ \Pi_{\mathsf{Bil}}}_{\log_k n \text{ times}}$$

is a reduction of knowledge from $\mathcal{R}_{\mathsf{Bil}(m,n)}$ to $\mathcal{R}(\mathsf{LF}_m)$ with $O(mn)$ prover and verifier time complexity and $O(k^2 \cdot \log_k n)$ communication complexity.

3.3.3 Instantiating Spaces

In practice, we are tasked with instantiating the underlying vector spaces and corresponding tensor product. This also instantiates the corresponding computational assumptions. Because $\mathbb{F}$ has multiplication built in, when considering the tensor product against vectors over the underlying field, $\otimes$ always corresponds to the outer product: For instance, $\mathbb{G}^m \otimes \mathbb{F}^n$ is equivalent to $\mathbb{G}^{mn}$. Thus, our remaining task is to instantiate $\mathbb{G}$, $\mathbb{H}$, and $\mathbb{G} \otimes \mathbb{H}$. We highlight two options.

- *Prime Order Groups:* We can set $\mathbb{G}$ to be a group of prime order p and set $\mathbb{H}$ to be the underlying field $\mathbb{F} = \mathbb{Z}_p$. In this setting, $\otimes$ corresponds to group scalar multiplication, and $\mathbb{G} \otimes \mathbb{H} \cong \mathbb{G}$. The corresponding computational assumption (if needed) corresponds to the discrete logarithm assumption.
- *Bilinear Groups:* We can set $\mathbb{G} = \mathbb{H}$ to be a symmetric bilinear group with target group $\mathbb{G}_\top$. In this case $\otimes$ corresponds to the pairing operation $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\top$, and $\mathbb{G} \otimes \mathbb{H} \cong \mathbb{G}_\top$. The corresponding computational assumption, (if needed) corresponds to the double-pairing assumption [11].

3.4 A Proof of Knowledge for NP

In this section, we develop a proof of knowledge for NP with logarithmic communication by leveraging our reductions of knowledge for linear algebraic statements. In particular, we first show that an NP-complete relation, $\mathcal{R}_{\mathsf{ACS}}$, can be encoded as a sequence of linear and bilinear forms constraints over the same commitment. We then develop helper reductions of knowledge that reduce the task of checking many linear and bilinear forms over the same commitment to a single linear and bilinear form. We then apply our reductions of knowledge for linear forms and bilinear forms.

Definition 3.13 (Algebraic Constraint System [100]). Consider group $\mathbb{G}$ and corresponding field $\mathbb{F}$ such that the bilinear relation assumption holds for $(\mathbb{G}, \mathbb{F})$ and $(\mathbb{G}, \mathbb{G})$.

We define the NP-complete algebraic constraint relation, $\mathcal{R}_{\mathsf{ACS}}$, characterized by n variables, $m = O(n)$ constraints, and ℓ inputs as follows. The public parameters consist of $G \in \mathbb{G}^n$. The statement consists of m sparse constraint matrices $M_1, \ldots, M_m \in \mathbb{F}^{n \times n}$ such that the total number of non-zero values in *all* matrices combined is $O(n)$, public inputs and outputs vector $X \in \mathbb{F}^\ell$, and witness commitment $\overline{Z} \in \mathbb{G}$. A witness vector $W \in \mathbb{F}^{n-\ell}$ is satisfying if for $Z = (X, W)$, $Z^\top M_i Z = 0$ for all $i \in [m]$, and $G(Z) = \overline{Z}$.

We can encode $\mathcal{R}_{\mathsf{ACS}}$ to tensor relations as follows: First, the verifier can check that $((G \oplus \delta_i, \overline{Z} \oplus X_i), Z) \in \mathcal{R}(\mathsf{LF}_n)$ for all $i \in [\ell]$ to ensure that Z contains public vector X. To check the commitment and constraints, it is sufficient for the verifier to check that the prover knows $Z_1, Z_2 \in \mathbb{F}^n$ such that $(G, (M_i, 1, \overline{Z}, \overline{Z}, 0), (Z_1, Z_2)) \in \mathcal{R}_{\mathsf{Bil}(n,n)}$ for all $i \in [m]$. The bilinear relation assumptions ensure that Z, Z_1 and Z_2 are equal.

Next, we leverage the fact that all linear form checks and all bilinear form checks are over the same commitment to reduce these checks. We formally capture the set of linear and bilinear form checks over the same commitment as the multiple linear and bilinear forms relations.

Definition 3.14 (Multiple Linear Forms). We define $\mathcal{R}_{\mathsf{MLF}(n,\ell)}$ such that

$$((G, (V_1, \ldots, V_\ell), (\sigma_1, \ldots, \sigma_\ell), \overline{Z}), Z) \in \mathcal{R}_{\mathsf{MLF}(n,\ell)}$$

if and only if

$$((G \oplus V_i, \overline{Z} \oplus \sigma_i), Z) \in \mathcal{R}(\mathsf{LF}_n)$$

for all i in $[\ell]$.

Definition 3.15 (Multiple Bilinear Forms). We define $\mathcal{R}_{\mathsf{MBil}(m,n,\ell)}$ such that

$$(G, ((M_1, \ldots, M_\ell), r, (\sigma_1, \ldots, \sigma_\ell), \overline{Z}_1, \overline{Z}_2), (Z_1, Z_2)) \in \mathcal{R}_{\mathsf{MBil}(m,n,\ell)}$$

if and only if

$$(G, (M_i, r, \overline{Z}_1, \overline{Z}_2, \sigma_i), (Z_1, Z_2)) \in \mathcal{R}_{\mathsf{Bil}(m,n)}$$

for all i in $[\ell]$.

With these relations, the above encoding can be captured as a reduction of knowledge in which the prover and verifier do not interact but rather take as input an $\mathcal{R}_{\mathsf{ACS}}$ statement-witness pair and output the corresponding tensor-based statements and witnesses in the multiple linear forms and bilinear forms relations. This step can be interpreted as a Levin reduction.

Lemma 3.7 (Encoding NP as Tensor Relations). There exists a reduction of knowledge from $\mathcal{R}_{\mathsf{ACS}(m,n,\ell)}$ to $\mathcal{R}_{\mathsf{MBil}(n,n,m)} \times \mathcal{R}_{\mathsf{MLF}(n,\ell)}$ with $O(n)$ prover and verifier complexity, and no communication.

Because all ℓ checks for $\mathcal{R}_{\mathsf{MLF}(n,\ell)}$ concern the same committed value, we observe that they can be batched into a single check for $\mathcal{R}(\mathsf{LF}_n)$ using a random linear combination. In particular, the verifier can send a random challenge $r \in \mathbb{F}$. Together the prover and

verifier can compute $V \leftarrow \sum_i V_i \cdot r^i$ and $\sigma \leftarrow \sum_i \sigma_i \cdot r^i$ and reduce to checking that the prover knows Z such that $((G \oplus V, \overline{Z} \oplus \sigma), Z) \in \mathcal{R}(\mathsf{LF}_n)$. Similarly, we can reduce multiple bilinear forms over the same commitment to a single bilinear form. Formally, we have the following reductions.

Lemma 3.8 (Linear Forms Batch Reduction). For $n, m, \ell \in \mathbb{N}$, there exists a reduction of knowledge from $\mathcal{R}_{\mathsf{MLF}(n,\ell)}$ to $\mathcal{R}_{\mathsf{LF}(n)}$ with $O(n\ell)$ prover and verifier time complexity, and $O(1)$ communication complexity.

Lemma 3.9 (Bilinear Forms Batch Reduction). For $n, m, \ell \in \mathbb{N}$, there exists a reduction of knowledge from $\mathcal{R}_{\mathsf{MBil}(m,n,\ell)}$ to $\mathcal{R}_{\mathsf{Bil}(m,n)}$ with $O(mn\ell)$ prover and verifier time complexity, and $O(1)$ communication complexity.

Putting everything together, we arrive at a proof of knowledge for NP.[3]

Corollary 3.3 (a proof of Knowledge for NP). Let Π_{encode} be the reduction of knowledge from $\mathcal{R}_{\mathsf{ACS}(n,m,\ell)}$ to $\mathcal{R}_{\mathsf{MBil}(n,n,m)} \times \mathcal{R}_{\mathsf{MLF}(n,\ell)}$ (Lemma 3.7). Let Π_{batchLF} be the batching scheme for linear forms (Lemma 3.8). Let Π_{batchBil} be the batching scheme for bilinear forms (Lemma 3.9). Let Π_{LF_n} be the proof of knowledge for $\mathcal{R}(\mathsf{LF}_n)$ with decomposition parameter k (Construction 3.1). Let $\Pi_{\mathsf{Bil}(n,n)}$ be the reduction of knowledge from $\mathcal{R}_{\mathsf{Bil}(n,n)}$ to $\mathcal{R}_{\mathsf{LF}(n)}$ with decomposition parameter k (Corollary 3.2). Let Π_{id} be the identity reduction of knowledge (i.e., the prover and verifier output their inputs). Let Π_{foldBool} be a 2-folding scheme for $\mathcal{R}_\top$ (i.e., the verifier outputs true if both its inputs are true). Then

$$\Pi_{\mathsf{foldBool}} \circ (\Pi_{\mathsf{id}} \times \Pi_{\mathsf{LF}_m}) \circ (\Pi_{\mathsf{LF}_n} \times \Pi_{\mathsf{Bil}(n,n)}) \circ (\Pi_{\mathsf{batchLF}} \times \Pi_{\mathsf{batchBil}}) \circ \Pi_{\mathsf{encode}}$$

is a proof of knowledge for $\mathcal{R}_{\mathsf{ACS}(n,m,\ell)}$ with $O(n)$ prover and verifier time complexity, and $O(k^2 \log_k n)$ communication complexity.

3.5 Recovering the Sum-Check Protocol

In this section, we show how to express the sum-check protocol as a tensor reduction over (linearized) polynomials. The following development provides a starting point for decomposing proofs of knowledge that rely on the sum-check protocol [59, 84, 124, 135, 136, 137, 139] as a sequence of reductions of knowledge. For example, the original proof system based on the sum-check protocol, proposed by Goldwasser, Kalai, and Rothblum [84], recursively interleaves two reductions: The first reduces a claim about layer i of a circuit into two claims about layer $i-1$ of a circuit (via the sum-check protocol). The second reduction folds two claims about layer $i-1$ into a single claim about layer $i-1$.

Bootle, Chiesa, and Sotiraki [37] show that a large class of split-and-fold techniques can be viewed as a special case of sum-check protocols over commitments, which they call sum-check arguments. We loosely show the converse of this result: That is, we show that tensor

[3]Critically, we have that binary relations (such as $\mathcal{R}_\top$) can be interpreted as ternary relations that ignore the public parameters and that the corresponding reductions can be defined with respect to arbitrary generators. This ensures that the requirements for composition are satisfied.

reductions, which can be interpreted as an abstracted folding technique, generalize sum-check protocols. Bootle et al. further show that sum-check arguments can be instantiated with any commitment scheme which satisfies a certain structural decomposability property, and thus show that sum-check arguments generalize folding techniques over prime-order groups, bilinear groups, and unknown-order groups. The following generalization lemma formally interprets these results as tensor reductions.

Our high level approach is as follows: First, we recall a simplified definition of the sum-check protocol. Next, we define a linearized sum-check protocol which represents running the tensor reduction on linearized multivariate polynomials decomposed as univariate polynomials. This effectively fixes the modules and decomposition rules necessary to fully specify the tensor reduction. Finally, we prove that a single step of the sum-check protocol is structurally equivalent to a single step of the linearized sum-check protocol.

We begin by recalling the sum-check protocol generalized to modules [37]. We make several simplifications for the sake of a more lucid presentation: First, we only define and consider a single recursive step of the sum-check and prove that it is structurally equivalent to a single recursive step of the tensor reduction instantiated over multivariate polynomials. Equivalence between the full sum-check protocol and the full recursive tensor reduction follows by induction. Second, we have the verifier immediately compute the statement polynomial in each recursive step, as opposed to deferring this computation until the end. The purpose of this modification is to avoid having to carry the randomness generated by both the tensor reduction and sum-check protocol throughout all the rounds in a global statement. Finally, we assume that both protocols use the standard monomial basis. Similar results hold for an arbitrary basis.

Definition 3.16 (Sum-Check Relation). Consider ring R, R-module V, and subset $H \subseteq \mathsf{R}$. The sum-check relation R_{SC}, characterized by the number of variables n, is defined to be

$$\mathcal{R}_{\mathsf{SC}}(n) = \left\{ ((P, \sigma), \bot) \;\middle|\; \begin{array}{l} P : \mathsf{R}^n \to V, \sigma \in V, \\ \sum_{x_1,\ldots,x_n \in H} P(x_1, \ldots, x_n) = \sigma. \end{array} \right\}$$

For notational simplicity, we omit $\bot$.

Construction 3.7 (Sum-check Protocol [37, 106]). Consider ring R, R-module V, and subset $H = \{h_1, \ldots, h_m\} \subseteq \mathsf{R}$. Suppose for some polynomial $P : \mathsf{R}^n \to V$ with degree $K - 1$ in each variable, and claimed sum $\sigma \in V$, the verifier would like to check

$$(P, \sigma) \in \mathcal{R}_{\mathsf{SC}}(n)$$

The prover sends to the verifier the degree $K - 1$ polynomial

$$p(X) = \sum_{x_1,\ldots,x_{n-1} \in H} P(x_1, \ldots, x_{n-1}, X).$$

The verifier checks

$$\sum_{x_n \in H} p(x_n) = \sigma.$$

The verifier then samples and sends α from a sampling set Q in R. The prover and verifier then compute

$$\sigma' \leftarrow p(\alpha)$$
$$P'(X_1, \ldots, X_{n-1}) \leftarrow P(X_1, \ldots, X_{n-1}, \alpha),$$

reducing the original task to the task of checking

$$(P', \sigma') \in \mathcal{R}_{\mathsf{SC}}(n-1).$$

Lemma 3.10 (Sum-check Protocol [37, 106]). The sum-check protocol is a reduction from $\mathcal{R}_{\mathsf{SC}}(n)$ to $\mathcal{R}_{\mathsf{SC}}(n-1)$.

Next we describe a linearized sum-check statement and a corresponding linearized sum-check protocol which leverages the tensor reduction.

Consider ring R, R-module V, and subset $H \subseteq \mathsf{R}$. Suppose for some polynomial $P : \mathsf{R}^n \to V$ with degree $K-1$ in each variable, and claimed sum $\boldsymbol{\sigma} \in V$, the verifier would like to check

$$\sum_{u_1, \ldots, u_n \in H} P(u_1, \ldots, u_n) = \boldsymbol{\sigma}. \tag{3.9}$$

By the universality of the tensor product (Definition 3.4), there exists tensor $\boldsymbol{P} \in V \otimes \bigotimes_{i \in [n]} \mathsf{R}^K$ such that $P = \boldsymbol{P} \circ \iota$ where ι is defined to be

$$\iota(u_1, \ldots, u_n) = \bigotimes_{j \in [n]} \bigoplus_{k \in [K_j]} u_j^k.$$

Because $\boldsymbol{P}$ is linear in its inputs, by letting

$$\boldsymbol{U} = \sum_{u_1, \ldots, u_n \in H} \iota(u_1, \ldots, u_n),$$

the verifier can check equation (3.9) by checking

$$\boldsymbol{P}(\boldsymbol{U}) = \boldsymbol{\sigma}.$$

This motivates defining the corresponding *linearized* sum-check relation.

Definition 3.17 (Linearized Sum-Check Relation). Consider ring R, R-module V, and subset $H \subseteq \mathsf{R}$. The linearized sum-check relation $\mathcal{R}_{\mathsf{LSC}}$, characterized by the number of variables n, is defined to be

$$\mathcal{R}_{\mathsf{LSC}}(n) = \left\{ ((\boldsymbol{P}, \boldsymbol{\sigma}), \perp) \;\middle|\; \begin{array}{l} \boldsymbol{P} \in V \otimes \bigotimes_{i \in [n]} \mathsf{R}^K, \boldsymbol{\sigma} \in V, \\ \boldsymbol{U} = \sum_{u_1, \ldots, u_n \in H} \iota(u_1, \ldots, u_n), \\ \boldsymbol{P}(\boldsymbol{U}) = \boldsymbol{\sigma} \end{array} \right\}$$

For notational simplicity, we omit $\perp$.

Construction 3.8 (Linearized Sum-Check Protocol). For arbitrary ring R, R-module V, subset $H \subseteq \mathsf{R}$, and degree bound K, we build a reduction for $(\mathcal{R}_{\mathsf{LSC}}(n), \mathcal{R}_{\mathsf{LSC}}(n-1))$. Let $\{\delta_1, \ldots, \delta_K\}$ represent a canonical basis for R^K and let $H = \{h_1, \ldots, h_m\}$. For $(\boldsymbol{P}, \boldsymbol{\sigma}) \in \mathsf{R}_{\mathsf{LSC}}(n)$, and $\boldsymbol{U} = \sum_{u_1 \ldots, u_n \in H} \iota(u_1, \ldots, u_n)$ we have that

$$\boldsymbol{P} = \sum_{i \in [K]} \boldsymbol{P}_i \otimes \delta_i$$

for some $(n-1)$-dimensional tensors $\boldsymbol{P}_i$, and

$$\boldsymbol{U} = \sum_{j \in [m]} \boldsymbol{U}' \otimes \boldsymbol{h}_j$$

where $\boldsymbol{U}' = \sum_{u_1, \ldots, u_{n-1} \in H} \iota(u_1, \ldots, u_{n-1})$, and $\boldsymbol{h}_j = (h_j^0, \ldots, h_j^{K-1})$. Applying the tensor reduction with respect to this decomposition reduces the verifier's task of checking the original check to the task of checking

$$\Big(\sum_i \alpha^i \delta_i\Big)\Big(\sum_j \beta^j \boldsymbol{h}_j\Big) = x$$

which the verifier checks immediately and

$$\Big(\sum_i \alpha^i \boldsymbol{P}_i\Big)\Big((\sum_j \beta^j)\boldsymbol{U}'\Big) = y$$

for $\alpha, \beta, x, y \in \mathsf{R}$ generated during the reduction. Thus, the verifier computes $\boldsymbol{P}' = \sum_i \alpha^i \boldsymbol{P}_i$ and $\boldsymbol{\sigma}' = y/(\sum_j \beta^j)$ and reduces the original check to the task of checking $(\boldsymbol{P}', \boldsymbol{\sigma}') \in \mathcal{R}_{\mathsf{LSC}}(n-1)$.

Lemma 3.11 (Linearized Sum-Check Protocol). The linearized sum-check protocol is a reduction from $\mathcal{R}_{\mathsf{LSC}}(n)$, to $\mathcal{R}_{\mathsf{LSC}}(n-1)$.

Proof. Completeness and soundness follow from Theorem 3.1. □

Given constructions for both the sum-check protocol and the linearized sum-check protocol, we can now prove that the two are structurally equivalent. To do so we will show that a single iteration of the sum-check protocol is equivalent to first linearizing the statement polynomials, running the linearized sum-check protocol and mapping the resulting statement back into the original space, and additionally show that the generated transcript from the linearized sum-check protocol can be used to recover the transcript produced by the standard sum-check protocol. It is important to note that we *cannot* show that the linearized sum-check protocol transcript is equivalent to the sum-check protocol transcript. This is because the tensor reduction transcript inherently contains more structural information, which must be thrown out to recover the sum-check protocol transcript.

Lemma 3.12 (Structural Correspondence). Let Π_{LSC} represent the linearized sum-check protocol and let Π_{SC} represent the sum-check protocol. Define the bijection Φ from a statement in $\mathcal{R}_{\mathsf{LSC}}$ to a statement in $\mathcal{R}_{\mathsf{SC}}$ as follows

$$\Phi((\boldsymbol{P}, \sigma)) = (P, \sigma)$$

where, given that $\boldsymbol{P} \in (\mathbb{F}^d)^n$ is an n-dimensional tensor, P is a polynomial that encodes the value at index $(j_1, \ldots, j_n)$ as the coefficient of term $x_1^{j_1} x_2^{j_2} \ldots x_n^{j_n}$. Then, given that Π_{LSC} and Π_{SC} are instantiated on the same verifier randomness, then the following diagram commutes

$$\begin{array}{ccc} \mathcal{R}_{\mathsf{LSC}}(n) & \xrightarrow{\Pi_{\mathsf{LSC}}} & \mathcal{R}_{\mathsf{LSC}}(n-1) \\ \Big\downarrow \Phi & & \Big\downarrow \Phi \\ \mathcal{R}_{\mathsf{SC}}(n) & \xrightarrow{\Pi_{\mathsf{SC}}} & \mathcal{R}_{\mathsf{SC}}(n-1) \end{array}$$

and there exists PPT simulator $\mathcal{S}$ that can simulate the interaction of Π_{SC} given oracle access to the interaction of Π_{LSC}.

Proof. Given a transcript of Π_{SC}, let the simulator produce a transcript of Π_{LSC} as follows

$$\mathcal{S}\Big(\Big\{r_{ij}, s_{ij} \Big| i \in [K], j \in [m]\Big\}, \alpha, \beta\Big) \mapsto \{r_{i1} \mid i \in [K]\}, \alpha$$

Consider arbitrary $(\boldsymbol{P}, \sigma) \in \mathcal{R}_{\mathsf{LSC}}(n)$. Let

$$\boldsymbol{P} = \boldsymbol{P}_1 \otimes \delta_1 + \boldsymbol{P}_2 \otimes \delta_2 + \ldots + \boldsymbol{P}_K \otimes \delta_K$$

Then, by linearity of Φ, we have

$$P(X_1, \ldots, X_n) = P_1(X_1, \ldots, X_{n-1}) \cdot X_n^0 + \ldots + P_K(X_1, \ldots, X_{n-1}) \cdot X_n^{K-1}$$

where $\Phi(\boldsymbol{P}) = P$ and $\Phi(\boldsymbol{P}_i) = P_i$. Moreover, for $\boldsymbol{h}_j = (h_j^0, \ldots, h_j^K)$, the Π_{LSC} prover sends as its first message

$$\{\boldsymbol{P}_i(\boldsymbol{U}'), \delta_i(\boldsymbol{h}_j) | i \in [K], j \in [m]\}$$

This means

$$\mathcal{S}(\{\boldsymbol{P}_i(\boldsymbol{U}'), \delta_i(\boldsymbol{h}_j) | i \in [K], j \in [m]\}) = \{\boldsymbol{P}_i(\boldsymbol{U}') | i \in [K]\}$$

which, under the monomial basis, is precisely equal to the coefficients of $p(X)$ sent by the Π_{SC} prover. Additionally, by assumption both Π_{LSC}, and Π_{SC} are initialized with the same verifier randomness. This means that the challenge α is identical in both transcripts. Therefore, we have that the simulator produces a transcript identical to the transcript produced by Π_{SC}.

Next, we observe that

$$\Phi(P') = \Phi\Big(\sum_i P_i \cdot \alpha^i\Big) = \sum_i \Phi(P_i) \cdot \alpha^i = \sum_i \boldsymbol{P}_i \cdot \alpha^i = \boldsymbol{P}'$$

Additionally, we observe that

$$\sigma' = \sum_{x_1,\ldots,x_n} P(x_1,\ldots,x_{n-1},\alpha) = \sum_i \alpha^i \boldsymbol{P}_i(\boldsymbol{U}') = \sum_{i,j} \alpha^i \beta^j \boldsymbol{P}_i(\boldsymbol{U}')/\Big(\sum_j \beta^j\Big) = \boldsymbol{\sigma}'$$

Therefore, we have also have that $\Phi(P', \sigma') = (\boldsymbol{P}', \boldsymbol{\sigma}')$. □

Chapter 4

Folding Schemes

This chapter contains joint work with Srinath Setty and Ioanna Tzialla [99, 101].

The basic step in our inner product argument is a 2-move reduction to a smaller [inner product] statement.

– Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit,
Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting

Recall that the sumcheck protocol [106] reduces the task of the checking the sum of evaluations of a polynomial over n variables into the task of checking the sum of evaluations of a simpler polynomial over $n-1$ variables. As we saw in Chapter 3, the sumcheck protocol does so by first reducing to the task of checking the sum of evaluations over *two* different polynomials over $n-1$ variables. The sumcheck protocol then *folds* the task of checking these two sums into the task of checking a single sum *of the same size* (i.e., over $n-1$ variables) by computing a random combination of the two sums.

This two-to-one folding reduction shows up again in a proof of inner-product due to Bootle, Cerulli, Chaidos, Groth, and Petit [36] (Inspired by Bayer and Groth [19]), where the prover splits an N-sized inner-product instance into two $N/2$-sized inner-product instances, and then the prover and the verifier interactively combine the two $N/2$-sized instances into a single $N/2$-sized instance. This pattern is now common in all major inner-product proof

systems [16, 17, 42, 46, 96, 102].

This two-to-one pattern has been recently generalized for more complex relations. For instance, as discussed in Chapter 1, *aggregation schemes* for polynomial commitments [35, 38] and *unbounded aggregation schemes* for linear-map vector commitments [49] reduce the task of checking proofs of several openings to a commitment to checking a proof of a single opening to a commitment. More recently, *accumulation schemes* [45] can be viewed as reducing the task of checking several proofs of knowledge and several accumulators to checking a single accumulator.

One of the major contributions of this thesis is to capture all protocols that follow this pattern under a unifying abstraction, which we refer to as *folding schemes*. Intuitively, a folding scheme for a relation $\mathcal{R}$ (e.g., the inner-product relation) is an interactive protocol, between a prover and a verifier, that reduces the task of checking two instances in $\mathcal{R}$ to the task of checking a single instance in $\mathcal{R}$.

Definition 4.1 (Folding Scheme). A folding scheme for $\mathcal{R}_1^\mu$ and $\mathcal{R}_2^\nu$ is a structured reduction of knowledge of type $\mathcal{R}_1^\mu \times \mathcal{R}_2^\nu \to \mathcal{R}_1$. We call a folding scheme of type $\mathcal{R} \times \mathcal{R} \to \mathcal{R}$ simply as a folding scheme for $\mathcal{R}$.

Given a general definition, another major contribution of this chapter is to provide a highly efficient folding scheme for an NP-complete relation, using a novel *relaxation* technique. In particular, an NP-complete relation requires constraints that are at least quadratic (to encode multiplication). As such, if we were to try to combine two satisfying assignments for an NP-complete relation using a random linear combination (as we would for linear relations such as the inner-product relation) we would be stuck with additional cross terms. We demonstrate that these cross terms can be carefully accounted for by relaxing an NP-complete relation with additional linear terms.

In this chapter, we provide efficient folding schemes for NP-complete relations, which enable efficient amortization by reducing the task of checking many instances to the task of checking a single instance. In Chapter 5, we will show that folding schemes for NP enable efficient *recursive* proof systems, which have applications in efficient zero-knowledge virtual machines. Since the introduction of folding schemes [101], the literature has both adopted the general definition [33, 97, 99, 115, 121, 130, 145, 146] and the underlying relaxation technique [41, 114]. This gives us preliminary evidence that the community is profitably employing the "proofs as maps" perspective in both theory and practice.

4.1 Preliminaries

We begin by overviewing both polynomials and cryptographic commitments, which are necessary to develop our folding schemes.

4.1.1 Polynomials and Low-Degree Extension

We write $\mathbb{F}^d[X_1, \ldots, X_n]$ to denote multivariate polynomials over field $\mathbb{F}$ in the variables $X_1, \ldots, X_n$ with degree bound d for each variable. We omit the superscript if there is no degree bound. We recall several facts about polynomials.

Definition 4.2 (Multilinear Polynomial). A multivariate polynomial is called a multilinear polynomial if the degree of the polynomial in each variable is at most one.

Definition 4.3 (Low-Degree Polynomial). A multivariate polynomial P over a finite field $\mathbb{F}$ is called low-degree polynomial if the degree d of P in each variable is exponentially smaller than $|\mathbb{F}|$ (i.e., $d = O(\log |\mathbb{F}|)$).

Definition 4.4 (Low-Degree Extension). Suppose $g : \{0,1\}^\ell \to \mathbb{F}$ is a function that maps ℓ-bit elements into an element of $\mathbb{F}$. A *polynomial extension* of g is a minimum-degree ℓ-variate polynomial in $\mathbb{F}[X_1, \ldots, X_\ell]$, denoted $\widetilde{g}$, such that $\widetilde{g}(x) = g(x)$ for all $x \in \{0,1\}^\ell$. A *multilinear* polynomial extension (MLE) is a low-degree polynomial extension where the extension is a multilinear polynomial (i.e., in $\mathbb{F}^1[X_1, \ldots, X_\ell]$). Given a function $g : \{0,1\}^\ell \to \mathbb{F}$, the multilinear extension of $\widetilde{g}$ is defined as follows.

$$\widetilde{g}(x_1, \ldots, x_\ell) = \sum_{x' \in \{0,1\}^\ell} \widetilde{\mathsf{eq}}(x, x') \cdot g(x')$$

where $\widetilde{\mathsf{eq}}$ is the multilinear extension of the function $\mathsf{eq} : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$, where $\mathsf{eq}(x, x') = 1$ if and only if $x = x'$. In particular, we can defined

$$\widetilde{\mathsf{eq}}(x, x') = \prod_{i=1}^{\ell} \left(x'_i \cdot x_i + (1 - x'_i) \cdot (1 - x_i)\right).$$

Definition 4.5 (Sparse Multilinear Polynomial). A multilinear polynomial in ℓ variables is a sparse multivariate polynomial if the number of non-zero evaluations over $\{0,1\}^\ell$ is sublinear in 2^ℓ.

4.1.2 Commitment Schemes

We define commitment schemes, which are a central building block for folding schemes.

Definition 4.6 (Commitment Scheme). A commitment scheme is defined by polynomial-time algorithm $\mathsf{Gen} : \mathbb{N}^2 \to P$ that produces public parameters given the security parameter and size parameter, a deterministic polynomial-time algorithm $\mathsf{Com} : P \times M \times R \to C$ that produces a commitment in C given a public parameters, message, and randomness tuple such that binding holds. That is, for any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda, n)$, and given $((m_1, r_1), (m_2, r_2)) \leftarrow \mathcal{A}(\mathsf{pp})$ we have that

$$\Pr[(m_1, r_1) \neq (m_2, r_2) \wedge \mathsf{Com}(\mathsf{pp}, m_1, r_1) = \mathsf{Com}(\mathsf{pp}, m_2, r_2)] \approx 0.$$

The commitment scheme is deterministic if Com does not use its randomness.

Definition 4.7 (Hiding). The commitment scheme $(\mathsf{Gen}, \mathsf{Com})$ is hiding if for any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda, n)$, $((m_1, r_1), (m_2, r_2)) \leftarrow \mathcal{A}(\mathsf{pp})$, and $C_i \leftarrow \mathsf{Com}(\mathsf{pp}, m_i, r_i)$ for $i \in \{1, 2\}$ we have that

$$\Pr[\mathcal{A}(\mathsf{pp}, C_1) = 1] \approx \Pr[\mathcal{A}(\mathsf{pp}, C_2) = 1].$$

Definition 4.8 (Homomorphic). The commitment scheme $(\mathsf{Gen}, \mathsf{Com})$ is homomorphic if the message space M, randomness space R, and commitment space C are groups and for all $n \in \mathbb{N}$, and $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda, n)$, we have that for any $m_1, m_2 \in M$ and $r_1, r_2 \in R$

$$\mathsf{Com}(\mathsf{pp}, m_1, r_1) + \mathsf{Com}(\mathsf{pp}, m_2, r_2) = \mathsf{Com}(\mathsf{pp}, m_1 + m_2, r_1 + r_2).$$

Definition 4.9 (Succinct Commitments). A commitment scheme $(\mathsf{Gen}, \mathsf{Com})$, over message space M and commitment space R, provides succinct commitments if for all $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$, and any $m \in M$ and $r \in R$, we have that $|\mathsf{Com}(\mathsf{pp}, m, r)| = O_\lambda(\mathsf{polylog}(|m|))$.

Definition 4.10 (Polynomial Commitment Scheme). A polynomial commitment scheme over polynomial ring $\mathbb{F}[X_1, \ldots, X_n]$ is a commitment scheme $(\mathsf{Gen}, \mathsf{Com})$ over message space $\mathbb{F}[X_1, \ldots, X_n]$, equipped with a proof of knowledge (Definition 2.7) for the relation $\mathcal{R}_{\mathsf{polyeval}}$ defined as follows

$$\mathcal{R}_{\mathsf{polyeval}} = \left\{ \; (\mathsf{pp}, (\overline{P}, x, y), (P, r)) \;\middle|\; \begin{array}{l} P \in \mathbb{F}[X_1, \ldots, X_n], \\ P(x) = y, \\ \overline{P} = \mathsf{Com}(\mathsf{pp}, P, r) \end{array} \right\}.$$

Given a formal notion of a commitment we can define generic relations over commitments. Below, we define the opening relation, where the instance is a commitment and the corresponding witness is a valid opening.

Definition 4.11 (Opening Relation). We define the knowledge of opening relation Open characterized by commitment scheme $\mathsf{com} = (\mathrm{gen}, \mathrm{com})$ as follows.

$$\mathsf{Open}_{\mathsf{com}} = \left\{ \; (\mathsf{pp}, \overline{w}, (w, r)) \;\middle|\; \overline{w} = \mathrm{com}(\mathsf{pp}, w, r) \; \right\}.$$

Given any relation, we can consider a variant where a commitment to the witness is additionally presented in the instance. We generically refer to such relations as *committed relations*.

Definition 4.12 (Committed Relation). Consider a relation $\mathcal{R}$ over structure, instance, witness tuples where witnesses are in some space W. Consider a commitment scheme $\mathsf{com} = (\mathsf{Gen}, \mathsf{Com})$ over message space W. We define the corresponding committed relation over public parameter, structure, instance, witness tuples characterized by com as follows.

$$\mathcal{R}(\mathsf{com}) = \left\{ \; (\mathsf{pp}_{\mathsf{com}}, \mathsf{s}, (C, u), (w, r)) \;\middle|\; \begin{array}{l} (\mathsf{s}, u, w) \in \mathcal{R}, \\ C = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, w, r) \end{array} \right\}$$

We say relation $\mathcal{R}$ is the underlying relation for committed relation $\mathcal{R}(\mathsf{com})$.

4.2 Folding Relaxed R1CS

In this section, we describe a public-coin, zero-knowledge succinct folding scheme for R1CS, a popular algebraic representation that generalizes arithmetic circuit satisfiability. We begin by recalling the definition of R1CS.

Definition 4.13 (R1CS). Consider a finite field $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$. The R1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. An instance $\mathsf{x} \in \mathbb{F}^{\ell}$ consists of public inputs and outputs and is satisfied by a witness $W \in \mathbb{F}^{m-\ell-1}$ if $(A \cdot Z) \circ (B \cdot Z) = C \cdot Z$, where $Z = (W, \mathsf{x}, 1)$.

As we show in Chapter 5, to realize IVC, we only need a folding scheme that can fold two R1CS instances with the same R1CS matrices (A, B, C). Specifically, given R1CS matrices (A, B, C), and two corresponding instance-witness pairs (x_1, W_1) and (x_2, W_2), we would like to devise a scheme that reduces the task of checking both instances into the task of checking a single new instance-witness pair (x, W) against the same R1CS matrices (A, B, C). Unfortunately, as we illustrate now, it is difficult to devise a folding scheme for R1CS such that it satisfies completeness, let alone knowledge soundness.

As R1CS is an algebraic system, the most direct approach would be to take a random linear combination. Ignoring efficiency concerns, suppose that the prover sends witnesses W_1 and W_2 in the first step. The verifier responds with a random $r \in \mathbb{F}$; the prover and the verifier both compute

$$\begin{aligned} \mathsf{x} &\leftarrow \mathsf{x}_1 + r \cdot \mathsf{x}_2 \\ W &\leftarrow W_1 + r \cdot W_2, \end{aligned}$$

and set the new instance-witness pair to be (x, W). However, for non-trivial $Z_1 = (W_1, \mathsf{x}_1, 1)$ and $Z_2 = (W_2, \mathsf{x}_2, 1)$, and $Z = (W, \mathsf{x}, 1)$, we *roughly* have that

$$\begin{aligned} AZ \circ BZ &= A(Z_1 + r \cdot Z_2) \circ B(Z_1 + r \cdot Z_2) \\ &= AZ_1 \circ BZ_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (AZ_2 \circ BZ_2) \\ &\neq CZ. \end{aligned}$$

The failed attempt exposes three issues. First, we must account for an additional cross-term, $r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1)$. Second, the terms excluding the cross-term combine to produce a term that does not equal CZ:

$$AZ_1 \circ BZ_1 + r^2 \cdot (AZ_2 \circ BZ_2) = CZ_1 + r^2 \cdot CZ_2 \neq CZ_1 + r \cdot CZ_2 = CZ.$$

Third, we do not even have that $Z = Z_1 + r \cdot Z_2$ because $Z_1 + r \cdot Z_2 = (W, \mathsf{x}, 1 + r \cdot 1)$.

To handle the first issue, we introduce a "slack" (or error) vector $E \in \mathbb{F}^m$ which absorbs the cross terms generated by folding. To handle the second and third issues, we introduce a scalar u, which absorbs an extra factor of r in $CZ_1 + r^2 \cdot CZ_2$ and in $Z = (W, \mathsf{x}, 1 + r \cdot 1)$. We refer to a variant of R1CS with these additional terms as *relaxed* R1CS.

Definition 4.14 (Relaxed R1CS). Consider a finite field $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$. The relaxed R1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. A relaxed R1CS instance consists of an error vector $E \in \mathbb{F}^m$, a scalar $u \in \mathbb{F}$, and public inputs and outputs $\mathsf{x} \in \mathbb{F}^{\ell}$. An instance (E, u, x) is satisfied by a witness $W \in \mathbb{F}^{m-\ell-1}$ if $(A \cdot Z) \circ (B \cdot Z) = u \cdot (C \cdot Z) + E$, where $Z = (W, \mathsf{x}, u)$.

Note that any R1CS instance can be expressed as a relaxed R1CS instance by augmenting it with $u = 1$ and $E = 0$, so relaxed R1CS retains NP-completeness.

Building on the first attempt, the prover and verifier can now use E to accumulate the cross-terms. In particular, for $Z_i = (W_i, \mathsf{x}_i, u_i)$, the prover and verifier additionally compute

$$\begin{aligned} u &\leftarrow u_1 + r \cdot u_2 \\ E &\leftarrow E_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1 - u_1 CZ_2 - u_2 CZ_1) + r^2 \cdot E_2, \end{aligned}$$

and set the new instance-witness pair to be $((E, u, \mathsf{x}), W)$. Conveniently, updating u in this manner also keeps track of how the constant term in Z should be updated, which motivates our choice to use u in $Z = (W, \mathsf{x}, u)$ rather than introducing a new variable. Now, for $Z = (W, \mathsf{x}, u)$, and for random $r \in \mathbb{F}$,

$$\begin{aligned} AZ \circ BZ &= AZ_1 \circ BZ_1 + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (AZ_2 \circ BZ_2) \\ &= (u_1 CZ_1 + E_1) + r \cdot (AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2 \cdot (u_2 CZ_2 + E_2) \\ &= (u_1 + r \cdot u_2) \cdot C(Z_1 + rZ_2) + E \\ &= uCZ + E. \end{aligned}$$

This implies that, for R1CS matrices (A, B, C), the folded witness W is a satisfying witness for the folded instance (E, u, x) as promised. A few issues remain: in the above scheme, the prover sends witnesses (W_1, W_2) for the verifier to compute E. As a result, the folding scheme is *not* non-trivial; it is also not zero-knowledge.

To circumvent these issues, we use succinct and hiding additively homomorphic commitments to W and E in the instance, and treat both W and E as the witness. We refer to this variant of relaxed R1CS as *committed relaxed R1CS*. Below, we describe a folding scheme for committed relaxed R1CS, where the prover sends a single commitment to aid the verifier in computing commitments to the folded witness (W, E).

Definition 4.15 (Committed Relaxed R1CS). Consider a finite field $\mathbb{F}$ and a commitment scheme Com over $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$, and commitment parameters pp_W and pp_E for vectors of size m and $m - \ell - 1$ respectively. The committed relaxed R1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. A committed relaxed R1CS instance is a tuple $(\overline{E}, u, \overline{W}, \mathsf{x})$, where $\overline{E}$ and $\overline{W}$ are commitments, $u \in \mathbb{F}$, and $\mathsf{x} \in \mathbb{F}^\ell$ are public inputs and outputs. An instance $(\overline{E}, u, \overline{W}, \mathsf{x})$ is satisfied by a witness $(E, r_E, W, r_W) \in (\mathbb{F}^m, \mathbb{F}, \mathbb{F}^{m-\ell-1}, \mathbb{F})$ if $\overline{E} = \mathsf{Com}(\mathsf{pp}_E, E, r_E)$, $\overline{W} = \mathsf{Com}(\mathsf{pp}_W, W, r_W)$, and $(A \cdot Z) \circ (B \cdot Z) = u \cdot (C \cdot Z) + E$, where $Z = (W, \mathsf{x}, u)$.

Construction 4.1 (A Folding Scheme for Committed Relaxed R1CS). Consider a finite field $\mathbb{F}$ and a succinct, hiding, and homomorphic commitment scheme Com over $\mathbb{F}$. We define the generator and the encoder as follows.

- $\mathcal{G}(1^\lambda, (m, n, \ell)) \to \mathsf{pp}$: output size bounds $m, n, \ell \in \mathbb{N}$, and commitment parameters pp_W and pp_E for vectors of size m and $m - \ell - 1$ respectively.

- $\mathcal{K}(\mathsf{pp}, (A, B, C)) \to (\mathsf{pk}, \mathsf{vk})$: output $\mathsf{pk} \leftarrow (\mathsf{pp}, (A, B, C))$ and $\mathsf{vk} \leftarrow \perp$.

The verifier $\mathcal{V}$ takes committed relaxed R1CS instances $(\overline{E}_1, u_1, \overline{W}_1, \mathsf{x}_1)$ and $(\overline{E}_2, u_2, \overline{W}_2, \mathsf{x}_2)$. The prover $\mathcal{P}$ additionally takes as input witnesses to both instances, $(E_1, r_{E_1}, W_1, r_{W_1})$ and $(E_2, r_{E_2}, W_2, r_{W_2})$. Let $Z_1 = (W_1, \mathsf{x}_1, u_1)$ and $Z_2 = (W_2, \mathsf{x}_2, u_2)$. The prover and the verifier proceed as follows.

1. $\mathcal{P}$: Send $\overline{T} \coloneqq \mathsf{Com}(\mathsf{pp}_E, T, r_T)$, where $r_T \xleftarrow{\$} \mathbb{F}$ and with cross term

$$T = AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1.$$

2. $\mathcal{V}$: Sample and send challenge $r \xleftarrow{\$} \mathbb{F}$.

3. $\mathcal{V}, \mathcal{P}$: Output the folded instance $(\overline{E}, u, \overline{W}, \mathsf{x})$ where

$$\begin{aligned}
\overline{E} &\leftarrow \overline{E}_1 + r \cdot \overline{T} + r^2 \cdot \overline{E}_2 \\
u &\leftarrow u_1 + r \cdot u_2 \\
\overline{W} &\leftarrow \overline{W}_1 + r \cdot \overline{W}_2 \\
\mathsf{x} &\leftarrow \mathsf{x}_1 + r \cdot \mathsf{x}_2
\end{aligned}$$

4. $\mathcal{P}$: Output the folded witness (E, r_E, W, r_W), where

$$\begin{aligned}
E &\leftarrow E_1 + r \cdot T + r^2 \cdot E_2 \\
r_E &\leftarrow r_{E_1} + r \cdot r_T + r^2 \cdot r_{E_2} \\
W &\leftarrow W_1 + r \cdot W_2 \\
r_W &\leftarrow r_{W_1} + r \cdot r_{W_2}
\end{aligned}$$

Theorem 4.1 (A Folding Scheme for Committed Relaxed R1CS). Construction 4.1 is a public-coin zero-knowledge, succinct folding scheme for committed relaxed R1CS.

Proof Intuition. With textbook algebra, we can show that if witnesses $(E_1, r_{E_1}, W_1, r_{W_1})$ and $(E_2, r_{E_2}, W_2, r_{W_2})$ are satisfying witnesses, then the folded witness (E, r_E, W, r_W) must be a satisfying witness. We prove knowledge soundness via the forking lemma (Lemma 2.2) by showing that the extractor can produce the initial witnesses given three accepting transcripts and the corresponding folded witnesses. Specifically, the extractor uses all three transcripts to compute E_i and r_{E_i}, and any two transcripts to compute W_i and r_{W_i} for $i \in \{1, 2\}$. The choice of which two transcripts does not matter due to the binding property of the commitment scheme. □

Lemma 4.1 (Completeness). Construction 4.1 satisfies completeness.

Proof. Consider a finite field $\mathbb{F}$, a succinct, hiding, and homomorphic commitment scheme Com over $\mathbb{F}$, and public parameters consisting of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$, and commitment parameters pp_W and pp_E for vectors of size m and $m - \ell - 1$ respectively sampled by the generator. Consider an adversarially chosen R1CS structure $(A, B, C) \in \mathbb{F}^{m \times m}$ and two committed relaxed R1CS instances

$$\varphi_1 = (\overline{E}_1, u_1, \overline{W}_1, \mathsf{x}_1) \quad \text{and} \quad \varphi_2 = (\overline{E}_2, u_2, \overline{W}_2, \mathsf{x}_2).$$

Suppose that the prover $\mathcal{P}$, in addition to the two instances, holds *satisfying* witnesses to both instances, $(E_1, r_{E_1}, W_1, r_{W_1})$ and $(E_2, r_{E_2}, W_2, r_{W_2})$. Let $Z_1 = (W_1, \mathsf{x}_1, u_1)$ and $Z_2 = (W_2, \mathsf{x}_2, u_2)$.

Now suppose that $\mathcal{P}$ and $\mathcal{V}$ compute a folded instance $\varphi = (\overline{E}, u, \overline{W}, \mathsf{x})$, and suppose that $\mathcal{P}$ computes a folded witness (E, r_E, W, r_W), To prove completeness, we must show that (E, W) is a satisfying witness for instance φ. Let $Z = (W, x, u)$.

For (E, W) to be a satisfying witness, we must have the following:

$$AZ \circ BZ = u \cdot CZ + E \tag{4.1}$$

and

$$\overline{E} = \mathsf{Com}(\mathsf{pp}_E, E, r_E) \tag{4.2}$$
$$\overline{W} = \mathsf{Com}(\mathsf{pp}_W, W, r_W) \tag{4.3}$$

It is easy to see that Equations (4.2) and (4.3) hold from the additive homomorphism of the commitment scheme.

Thus, we focus on proving that Equation (4.1) holds. By construction, for Equation (4.1) to hold, we must have for $r \in_R \mathbb{F}$

$$A(Z_1 + r \cdot Z_2) \circ B(Z_1 + r \cdot Z_2) = (u_1 + r \cdot u_2) \cdot C(Z_1 + r \cdot Z_2) + E.$$

Distributing, we must have

$$\begin{aligned} &AZ_1 \circ BZ_1 + r(AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1) + r^2(AZ_2 \circ BZ_2) = \\ &u_1 \cdot CZ_1 + r(u_1 \cdot CZ_2 + u_2 CZ_1) + r^2 \cdot u_2 \cdot CZ_2 + E. \end{aligned}$$

Aggregating by powers of r, we must have

$$\begin{aligned} &(AZ_1 \circ BZ_1 - u_1 \cdot CZ_1) + \\ &r(AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1 - u_1 \cdot CZ_2 - u_2 CZ_1) + \\ &r^2(AZ_2 \circ BZ_2 - u_2 \cdot CZ_2) \\ =&E. \end{aligned} \tag{4.4}$$

However, because W_1 and W_2 are satisfying witnesses, we have

$$\begin{aligned} AZ_1 \circ BZ_1 - u_1 \cdot CZ_1 &= E_1 \\ AZ_2 \circ AZ_2 - u_2 \cdot CZ_2 &= E_2. \end{aligned}$$

Additionally, by construction we have

$$AZ_1 \circ BZ_2 + AZ_2 \circ BZ_1 - u_1 \cdot CZ_2 - u_2 \cdot CZ_1 = T.$$

Thus, by substitution, for Equation (4.4) to hold we must have

$$E_1 + r \cdot T + r^2 \cdot E_2 = E,$$

which holds by construction.

□

Lemma 4.2 (Knowledge Soundness). Construction 4.1 satisfies knowledge soundness.

Proof. Consider a finite field $\mathbb{F}$, a succinct, hiding, and homomorphic commitment scheme Com over $\mathbb{F}$, and public parameters consisting of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$, and commitment parameters pp_W and pp_E for vectors of size m and $m - \ell - 1$ respectively sampled by the generator. Consider an adversarially chosen R1CS structure $(A, B, C) \in \mathbb{F}^{m \times m}$ and two committed relaxed R1CS instances

$$\varphi_1 = (\overline{E}_1, u_1, \overline{W}_1, \mathsf{x}_1) \quad \text{and} \quad \varphi_2 = (\overline{E}_2, u_2, \overline{W}_2, \mathsf{x}_2).$$

We prove knowledge soundness via tree extraction (Lemma 2.2). That is, we prove that there exists a PPT algorithm $\mathcal{X}$ such that when given public parameters pp, structure (A, B, C), and a tree of accepting transcripts and the corresponding folded instance-witness pairs outputs a satisfying witness with probability $1 - \mathsf{negl}(\lambda)$.

In more detail, suppose $\mathcal{X}$ is provided three transcripts (τ_1, τ_2, τ_3) with the same initial commitment $\overline{T}$ from the prover. Note that a transcript τ_i for $i \in \{1, 2, 3\}$ additionally comes attached with an accepting witness $\tau_i.(W, E, r_W, r_E)$ and the verifier's randomness $\tau_i.r$. Interpolating points $(\tau_1.r, \tau_1.W)$ and $(\tau_2.r, \tau_2.W)$, $\mathcal{X}$ retrieves (W_1, W_2) such that

$$W_1 + \tau_i.r \cdot W_2 = \tau_i.W \tag{4.5}$$

for $i \in \{1, 2\}$. Similarly interpolating points $(\tau_1.r, \tau_1.E)$, $(\tau_2.r, \tau_2.E)$, $(\tau_3.r, \tau_3.E)$, $\mathcal{X}$ retrieves (E_1, E_2) and a cross-term T such that

$$E_1 + \tau_i.r \cdot T + \tau_i.r^2 \cdot E_2 = \tau_i.E \tag{4.6}$$

for $i \in \{1, 2, 3\}$. Using the same approach, $\mathcal{X}$ can interpolate for r_{W_1}, r_{W_2} and r_{E_1}, r_T, r_{E_2}. We must argue that $(W_1, E_1, r_{W_1}, r_{E_1})$ and $(W_2, E_2, r_{W_2}, r_{E_2})$ are indeed satisfying witnesses for φ_1 and φ_2 respectively.

We first show that the retrieved witness elements are valid openings to the corresponding commitments in the instance. For $i \in \{1, 2\}$, because $\tau_i.W, \tau_i.r_W$ is part of a satisfying witness, by construction,

$$\begin{aligned}
&\mathsf{Com}(\mathsf{pp}_W, W_1, r_{W_1}) + \tau_i.r \cdot \mathsf{Com}(\mathsf{pp}_W, W_2, r_{W_2}) \\
&= \mathsf{Com}(\mathsf{pp}_W, W_1 + \tau_i.r \cdot W_2, r_{W_1} + \tau_i.r \cdot r_{W_2}) \\
&= \mathsf{Com}(\mathsf{pp}_W, \tau_i.W, \tau_i.r_W)
\end{aligned}$$

$$= \overline{W}_1 + \tau_i.r \cdot \overline{W}_2.$$

Interpolating, we must have that

$$\mathsf{Com}(\mathsf{pp}_W, W_1, r_{W_1}) = \overline{W}_1 \tag{4.7}$$
$$\mathsf{Com}(\mathsf{pp}_W, W_2, r_{W_2}) = \overline{W}_2. \tag{4.8}$$

Similarly, for $i \in \{1, 2, 3\}$, because $\tau_i.E, \tau_i.r_E$ is part of a satisfying witness, by construction, we must have

$$\begin{aligned}
&\mathsf{Com}(\mathsf{pp}_E, E_1, r_{E_1}) + \tau_i.r \cdot \mathsf{Com}(\mathsf{pp}_E, T, r_T) + \tau_i.r^2 \cdot \mathsf{Com}(\mathsf{pp}_E, E_2, r_{E_2}) \\
&= \mathsf{Com}(\mathsf{pp}_E, E_1 + \tau_i.r \cdot T + \tau_i.r^2 \cdot E_2, r_{E_1} + \tau_i.r \cdot r_T + \tau_i.r^2 \cdot r_{E_2}) \\
&= \mathsf{Com}(\mathsf{pp}_E, \tau_i.E, \tau_i.r_E) \\
&= \overline{E}_1 + \tau_i.r \cdot \overline{T} + \tau_i.r^2 \cdot \overline{E}_2.
\end{aligned}$$

Interpolating, we must have that

$$\mathsf{Com}(\mathsf{pp}_E, E_1, r_{E_1}) = \overline{E}_1$$
$$\mathsf{Com}(\mathsf{pp}_E, E_2, r_{E_2}) = \overline{E}_2$$

Next, we must show that (W_1, E_1) and (W_2, E_2) satisfy the relaxed R1CS relation. To show this, we must first argue that Equation (4.5) holds for $i = 3$ as well (i.e., $W_1 + \tau_3.r \cdot W_2 = \tau_3.W$). Indeed, Equations (4.7) and (4.8) imply that

$$\begin{aligned}
&\mathsf{Com}(\mathsf{pp}_W, W_1 + \tau_3.r \cdot W_2, r_{W_1} + \tau_3.r \cdot r_{W_2}) \\
&= \mathsf{Com}(\mathsf{pp}_W, W_1, r_{W_1}) + \tau_3.r \cdot \mathsf{Com}(\mathsf{pp}_W, W_2, r_{W_2}) \\
&= \overline{W}_1 + \tau_3.r \cdot \overline{W}_2 \\
&= \mathsf{Com}(\mathsf{pp}_W, \tau_3.W, \tau_3.W)
\end{aligned}$$

Thus, by the binding property of Com, we must additionally have that

$$W_1 + \tau_3.r \cdot W_2 = \tau_3.W \tag{4.9}$$

with probability $1 - \mathsf{negl}(\lambda)$.

Because $(\tau_i.W, \tau_i.E)$ is part of a satisfying witness, for $i \in \{1, 2, 3\}$ we have

$$A(\tau_i.Z) \circ B(\tau_i.Z) = u \cdot C(\tau_i.Z) + \tau_i.E$$

where $\tau_i.Z = (\tau_i.W, \tau_i.x, \tau_i.u)$. However, by Equations (4.5), (4.6), and (4.9) for $i \in \{1, 2, 3\}$, this implies that with probability $1 - \mathsf{negl}(\lambda)$

$$\begin{aligned}
&A(Z_1 + \tau_i.r \cdot Z_2) \circ B(Z_1 + \tau_i.r \cdot Z_2) = \\
&(u_1 + \tau_i.r \cdot u_2) \cdot C(Z_1 + \tau_i.r \cdot Z_2) + (E_1 + \tau_i.r \cdot T + \tau_i.r^2 \cdot E_2)
\end{aligned}$$

where $Z_1 = (W_1, x_1, u_1)$ and $Z_2 = (W_2, x_2, u_2)$. Expanding and interpolating, we have that

$$AZ_1 \circ BZ_1 = u_1 \cdot CZ_1 + E_1$$

$$AZ_2 \circ BZ_2 = u_2 \cdot CZ_2 + E_2$$

with probability $1 - \mathsf{negl}(\lambda)$. Thus, $(W_1, E_1, r_{W_1}, r_{E_1})$ and $(W_2, E_2, r_{W_2}, r_{E_2})$ meet all the requirements to be satisfying witnesses for φ_1 and φ_2 respectively with probability $1 - \mathsf{negl}(\lambda)$. □

Lemma 4.3 (Zero-Knowledge). Construction 4.1 satisfies zero-knowledge.

Proof. Intuitively, zero-knowledge holds because the prover only sends a single hiding commitment. More formally, the simulator $\mathcal{S}$ samples random $T \in \mathbb{F}^m$ and $r \in \mathbb{F}$ and computes $\overline{T} = \mathsf{Com}(\mathsf{pp}_E, T, r)$. Next, $\mathcal{S}$ derives the verifier's challenge, r, using ρ and outputs $\mathsf{tr} = (\overline{T}, r)$. (Perfect) zero-knowledge holds from the (perfect) hiding property of the underlying commitment scheme. □

Lemma 4.4 (Efficiency). Construction 4.1 induces a folding scheme from R1CS and committed relaxed R1CS to committed relaxed R1CS where the verifier's time complexity is dominated by two group scalar multiplications.

Proof. The prover and verifier can embed an R1CS instance as a committed relaxed R1CS instance by setting $(E, r_E, u) = (\mathbf{0}, 0, 1)$ and setting the commitments accordingly. As such, given that the second input is an R1CS instance (as opposed to a committed relaxed R1CS instance) we have that $\overline{E}_2$ is a commitment to the zero vector. Thus, the verifier does not need to compute $r^2 \cdot \overline{E}_2$. □

4.3 Folding Customizable Constraint Systems

Often, in practice, computations are more efficiently expressed with high-degree constraints. As a concrete example, consider the MinRoot function, which is intentionally expensive to compute, and thus can be used as a building block for a proof-of-work mechanism [93]. In particular, for finite field $\mathbb{F}$ of prime order p, MinRoot is defined as follows

$$\mathsf{MinRoot}(x_i, y_i, i) = ((x_i + y_i)^{2p-1/3}, x_i, i + 1).$$

MinRoot can be represented with *one* degree-5 constraint in Plonkish [143]. Alternatively, MinRoot requires three constraints to be expressed in R1CS (as defined in the previous section), which is limited to quadratic constraints.

In practice, we must construct tailored zkSNARKs, such as Plonk [73], to prove higher-degree constraint systems such as Plonkish, where constraints are represented as multivariate polynomials. Our goal in this section is to design a folding scheme that similarly supports high-degree constraints. We will show in Chapter 5 that such a folding scheme in turn affords an IVC scheme that supports high-degree constraints.

As a starting point, Mohnblatt proposes Sangria [114], which adapts the folding scheme for R1CS (Section 4.2) to support high-degree Plonkish constraints. However, this is achieved by using a similar error term strategy as the folding scheme for R1CS. Unfortunately, this means that the number of cross-terms that the prover must commit to increases

linearly with the degree of the constraints d: For n constraints, this incurs $O(n \cdot d)$ cryptographic operations to commit to $O(d)$ cross-terms, where n is the number of constraints.

In this section, we demonstrate that we can build a folding scheme for high-degree constraints where the prover's cryptographic work is independent of the degree of constraints supported. In particular, the number of multi-scalar-multiplications (MSMs) and their sizes will be independent of the degree of constraints supported. To do so, we build a folding scheme for CCS [127], a customizable constraint system that simultaneously generalizes Plonkish [73], R1CS, and the AIR constraint system due to Ben-Sasson et al. [28].

Definition 4.16 (CCS [127]). Consider size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ where $n > \ell$. Let $s = \log m$ and $s' = \log n$. We define the customizable constraint system (CCS) relation, $\mathcal{R}_{\mathsf{CCS}}$, over structure, instance, witness tuples as follows.

An $\mathcal{R}_{\mathsf{CCS}}$ structure s consists of

- a sequence of sparse multilinear polynomials in $s + s'$ variables $\widetilde{M}_1, \ldots, \widetilde{M}_t$ such that they evaluate to a non-zero value in at most $N = \Omega(m)$ locations over the Boolean hypercube $\{0,1\}^s \times \{0,1\}^{s'}$;
- a sequence of q multisets $[S_1, \ldots, S_q]$, where an element in each multiset is from the domain $\{1, \ldots, t\}$ and the cardinality of each multiset is at most d.
- a sequence of q constants $[c_1, \ldots, c_q]$, where each constant is from $\mathbb{F}$.

An $\mathcal{R}_{\mathsf{CCS}}$ instance consists of public input and output vector $\mathsf{x} \in \mathbb{F}^\ell$. An $\mathcal{R}_{\mathsf{CCS}}$ witness consists of a multilinear polynomial $\widetilde{w}$ in $s' - 1$ variables. We have that $(\mathsf{s}, \mathsf{x}, \widetilde{w}) \in \mathcal{R}_{\mathsf{CCS}}$ if and only if for all $x \in \{0,1\}^s$,

$$\sum_{i=1}^{q} c_i \cdot \left(\prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{\log m}} \widetilde{M}_j(x, y) \cdot \widetilde{z}(y) \right) \right) = 0,$$

where $\widetilde{z}$ is an s'-variate multilinear polynomial such that $\widetilde{z}(x) = \widetilde{(w, 1, \mathsf{x})}(x)$ for all $x \in \{0,1\}^{s'}$.

To fold CCS, our starting point is the observation that Spartan [124] (more specifically its generalization to handle CCS called SuperSpartan [127]) transforms the task of checking the satisfiability of a CCS instance into the task of checking if a multivariate polynomial g of total degree $d + 1$, where d is the degree of the CCS constraints, sums to zero over a suitable Boolean hypercube. Spartan then invokes the sum-check protocol [106] to prove that claim about g. At the end of the sum-check invocation, the prover and the verifier are left with checking certain claims. Fortunately, these claims concern a restricted form of CCS that we refer to as *linearized CCS*, which only contains linear constraints.

Definition 4.17 (Linearized CCS). Consider size bounds $m, n, N, \ell, t, q, d \in \mathbb{N}$ where $n = 2 \cdot (\ell + 1)$. Let $s = \log m$ and $s' = \log n$. We define the linearized committed customizable constraint system (LCCS) relation, $\mathcal{R}_{\mathsf{LCCS}}$, over structure, instance, witness tuples as follows.

An $\mathcal{R}_{\mathsf{LCCS}}$ structure s consists of

- a sequence of sparse multilinear polynomials in $s + s'$ variables $\widetilde{M}_1, \ldots, \widetilde{M}_t$ such that they evaluate to a non-zero value in at most $N = \Omega(m)$ locations over the Boolean hypercube $\{0,1\}^s \times \{0,1\}^{s'}$;
- a sequence of q multisets $[S_1, \ldots, S_q]$, where an element in each multiset is from the domain $\{1, \ldots, t\}$ and the cardinality of each multiset is at most d.
- a sequence of q constants $[c_1, \ldots, c_q]$, where each constant is from $\mathbb{F}$.

An $\mathcal{R}_{\mathsf{LCCS}}$ instance is a tuple $(u, \mathsf{x}, r, v_1, \ldots, v_t) \in (\mathbb{F}, \mathbb{F}^\ell, \mathbb{F}, \mathbb{F}^t)$. An $\mathcal{R}_{\mathsf{LCCS}}$ witness consists of a multilinear polynomial $\widetilde{w}$ in $s' - 1$ variables. We have that $(\mathsf{s}, (u, \mathsf{x}, r, v_1, \ldots, v_t), \widetilde{w}) \in \mathcal{R}_{\mathsf{LCCS}}$ if and only if for all $i \in [t]$

$$v_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r, y) \cdot \widetilde{z}(y)$$

where $\widetilde{z}$ is an s'-variate multilinear polynomial such that $z(x) = \widetilde{(w, u, \mathsf{x})}(x)$ for all $x \in \{0,1\}^{s'}$.

Spartan, being a full proof of knowledge, continues to prove the linearized CCS with an additional invocation of the sum-check protocol. In our case however, we can use the first portion of Spartan to reduce CCS to linearized CCS, and then naturally fold it into a running linearized CCS instance using a random linear combination. We can do this in one shot by redefining the polynomial g to additionally include claims from a running linearized CCS instance using a random challenge from the verifier. This is possible as long as the running instance and the CCS instance that is being folded share a *compatible* structure (e.g., the same CCS matrices). To ensure soundness, we work with variants of CCS and linearized CCS with commitments to the witness in the instance (Definition 4.12), which we denote as $\mathcal{R}_{\mathsf{CCCS}}$ and $\mathcal{R}_{\mathsf{LCCCS}}$ respectively.

4.3.1 Overview

As a warmup, we overview a folding scheme from $\mathcal{R}_{\mathsf{CCCS}}$ and $\mathcal{R}_{\mathsf{LCCCS}}$ to $\mathcal{R}_{\mathsf{CCCS}}$ in this section. In the next section, we provide a formal folding scheme of type $\mathcal{R}^{\mu}_{\mathsf{LCCCS}} \times \mathcal{R}^{\nu}_{\mathsf{CCCS}} \to \mathcal{R}_{\mathsf{LCCCS}}$ for arbitrary μ and ν.

Consider the CCS structure

$$\mathsf{s} = ((\widetilde{M}_1, \ldots, \widetilde{M}_t), (S_1, \ldots, S_q), (c_1, \ldots, c_q)),$$

and let $s = \log m$, and $s' = \log n$. We design a multi-folding scheme that reduces the verifier's task of checking a linearized committed CCS instance $(C_1, u, \mathsf{x}_1, r_x, v_1, \ldots, v_t)$ and a committed CCS instance (C_2, x_2) to the task of checking a new linearized committed CCS instance. In particular, the verifier's goal is to reduce the task of checking that a prover knows satisfying witnesses $\widetilde{w_1}$ and $\widetilde{w_2}$ such that for $\widetilde{z_1} = \widetilde{(w_1, u, \mathsf{x}_1)}$ and $\widetilde{z_2} = \widetilde{(w_2, 1, \mathsf{x}_2)}$, we have that

$$v_j = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r_x, y) \cdot \widetilde{z_1}(y) \tag{4.10}$$

for all $j \in [t]$ and

$$\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^s} \widetilde{M}_j(x, y) \cdot \widetilde{z}_2(y) \right) = 0 \tag{4.11}$$

for all $x \in \{0,1\}^s$.

The high-level strategy of the prover and verifier is to first encode the above claims as a claim about the evaluations of polynomials and then reduce this claim using the sum-check protocol. The resulting reduced claim is equivalent to checking two *compatible* linearized committed CCS instances. The compatibility ensures that we can reduce the task of checking both instances into the task of checking a single linearized CCS instance using a random linear combination.

In more detail, consider multilinear polynomials

$$H_j(x) \coloneqq \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_1(y) \tag{4.12}$$

and

$$G(x) \coloneqq \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_2(y) \right). \tag{4.13}$$

Then, checking $H_j(r_x) = v_j$ for all $j \in [t]$ implies checking Equation 4.10. Similarly, checking $G(x) = 0$ for all $x \in \{0,1\}^s$ implies checking Equation 4.11. To achieve succinctness we would now like to efficiently reduce all polynomial evaluation checks into a single check. The prover and verifier cannot simply take a random linear combination as the polynomials are not evaluated over the same domain. Instead, we recast the polynomial evaluation claims as claims regarding the sums of evaluations (over a fixed domain) for these polynomials. To do this, we leverage the following lemma.

Lemma 4.5 (Sums over Evaluations). Consider size $\ell \in \mathbb{N}$. For multilinear polynomial $P \in \mathbb{F}[X_1, \ldots, X_\ell]$ we have that

$$P(X) = \sum_{x \in \{0,1\}^\ell} \widetilde{\mathsf{eq}}(X, x) \cdot P(x).$$

where $\widetilde{\mathsf{eq}}$ is a multilinear extension of eq, which takes as inputs two values in $\{0,1\}^\ell$ returns 1 if its inputs are equal and 0 otherwise.

Proof. Let $Q(X) = \sum_{x \in \{0,1\}^\ell} \widetilde{\mathsf{eq}}(X, x) \cdot P(x)$. By the definition of $\widetilde{\mathsf{eq}}$, we have that

$$P(x) = Q(x)$$

for all $x \in \{0,1\}^\ell$. However, because $P \in \mathbb{F}[X_1, \ldots, X_\ell]$ is multilinear, it is completely determined by 2^ℓ evaluation points. The same holds for Q. Because P and Q agree on 2^ℓ points, they must be the same polynomial. □

Then, by Lemma 4.5, for $L_j(x) = \widetilde{\mathsf{eq}}(r_x, x) \cdot H_j(x)$, checking $H_j(r_x) = v_j$ for all $j \in [t]$ is equivalent to checking

$$v_j = \sum_{x \in \{0,1\}^s} L_j(x) \tag{4.14}$$

for all $j \in [t]$.

We define a corresponding Lagrange polynomial, $\sum_{x \in \{0,1\}^s} \widetilde{\mathsf{eq}}(X, x) \cdot G(x)$, which encodes each evaluation of G into its coefficients. Then checking that this Lagrange polynomial is the zero polynomial implies checking that $G(x) = 0$ for all $x \in \{0,1\}^s$. We can reduce the task of checking all 2^s evaluations to the task of checking a single evaluation by invoking the Schwartz-Zippel lemma.

Lemma 4.6 (Schwartz-Zippel [123]). Let $g : \mathbb{F}^\ell \to \mathbb{F}$ be an ℓ-variate non-zero polynomial of total degree at most d. Then, on any finite set $S \subseteq \mathbb{F}$,

$$\Pr_{x \xleftarrow{\$} S^\ell} [g(x) = 0] \leq d/|S|.$$

Then, for a random challenge $\beta \in \mathbb{F}$, by the Schwartz-Zippel Lemma, for $Q(x) = \widetilde{\mathsf{eq}}(\beta, x) \cdot G(x)$, checking

$$0 = \sum_{x \in \{0,1\}^s} Q(x) \tag{4.15}$$

implies checking Equation 4.11 with high probability.

Equations 4.14 and 4.15 can be checked simultaneously with high probability by setting

$$g(x) \leftarrow \left(\sum_{j \in [t]} \gamma^j \cdot L_j(x) \right) + \gamma^{t+1} \cdot Q(x)$$
$$T \leftarrow \left(\sum_{j \in T} \gamma^j \cdot v_j \right) + \gamma^{t+1} \cdot 0$$

for some random challenge $\gamma \in \mathbb{F}$ and checking

$$T = \sum_{x \in \{0,1\}^s} g(x). \tag{4.16}$$

Then, the prover and verifier run the sum-check protocol (Construction 4.3) to reduce the task of checking Equation 4.16 to the task of checking

$$c = g(r'_x) \tag{4.17}$$

for some random point $r'_x \in \mathbb{F}^s$ and claimed evaluation $c \in \mathbb{F}$.

To assist the verifier in checking Equation 4.17, the prover computes claimed values for sums internal to polynomial g,

$$\sigma_i \leftarrow \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r'_x, y) \cdot \widetilde{z}_1(y) \tag{4.18}$$

$$\theta_i \leftarrow \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r'_x, y) \cdot \widetilde{z}_2(y), \tag{4.19}$$

for all $i \in [t]$, and sends them to the verifier.

Using these values, the verifier can check Equation 4.17. However, it must still check Equations 4.18 and 4.19, that is, that σ_i and θ_i were computed correctly for all $i \in [t]$. We observe now that because both of these equations are defined with respect to the same sum-check randomness r'_x, by linearity, the verifier can sample a random challenge ρ, and reduce the task of checking Equations 4.18 and 4.19 to the task of checking

$$\sigma_i + \rho \cdot \theta_i = \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_i(r'_x, y) \cdot (\widetilde{z}_1(y) + \rho \cdot \widetilde{z}_2(y)) \tag{4.20}$$

for all $i \in [t]$.

Conveniently, letting $C' \leftarrow C_1 + \rho \cdot C_2$, $u' \leftarrow u + \rho \cdot 1$, $\mathsf{x}' \leftarrow \mathsf{x}_1 + \rho \cdot \mathsf{x}_2$, and $v'_i \leftarrow \sigma_i + \rho \cdot \theta_i$ for all $i \in [t]$, checking Equation 4.20, is equivalent to checking that the prover knows a witness for the following linearized committed CCS instance

$$(C', u', \mathsf{x}', r'_x, v'_1, \ldots, v'_t)$$

thus completing the reduction.

4.3.2 Construction

We now formally describe a folding scheme of type $\mathcal{R}^{\mu}_{\mathsf{LCCCS}} \times \mathcal{R}^{\nu}_{\mathsf{CCCS}} \to \mathcal{R}_{\mathsf{LCCCS}}$, which we describe modularly in three reductions: First, we present a reduction from $\mathcal{R}^{\mu}_{\mathsf{LCCCS}} \times \mathcal{R}^{\nu}_{\mathsf{CCCS}}$ to the *committed* sumcheck relation, $\mathcal{R}_{\mathsf{CSC}}$, which augments the original sumcheck relation (Definition 3.16) with a commitment to the underlying polynomial. We refer to this reduction as a preprocessing step (Construction 4.2). Formally, we define the committed sumcheck relation as follows.

Definition 4.18 (Committed Sumcheck Relation). We define the committed sumcheck relation $\mathcal{R}_{\mathsf{CSC}}(m)$ over public parameter, instance, witness pairs characterized by polynomial ring $\mathbb{F}[X_1, \ldots, X_n]$, and commitment scheme $(\mathsf{Gen}, \mathsf{Com})$ as follows

$$\mathcal{R}_{\mathsf{CSC}}(m) = \left\{ \begin{array}{l} \mathsf{pp}, \\ (\overline{g}, T, (r_{m+1}, \ldots, r_n)), \\ (g, s) \end{array} \middle| \begin{array}{l} g \in \mathbb{F}[X_1, \ldots, X_n], \\ T = \sum_{x_1, \ldots, x_m \in \{0,1\}} g(x_1, \ldots, x_m, r_{m+1} \ldots, r_n), \\ \overline{g} = \mathsf{Com}(\mathsf{pp}, g, s) \end{array} \right\}$$

Next, we formally describe the sumcheck protocol, recasting it as a as a reduction of knowledge from the committed sumcheck relation to the polynomial evaluation relation (Definition 4.10). Finally, we describe a reduction of knowledge from the polynomial evaluation relation (with respect to the specialized commitment scheme we describe) to the linearized CCS relation, completing the reduction.

Below we describe the preprocessing reduction from an arbitrary number of linearized CCS and CCS instances to a sumcheck instance.

Construction 4.2 (Folding CCS, Preprocessing). Let $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit})$ denote an additively-homomorphic commitment scheme for multilinear polynomials. We construct a reduction of knowledge of type $\mathcal{R}^{\mu}_{\mathsf{LCCCS}} \times \mathcal{R}^{\nu}_{\mathsf{CCCS}} \to \mathcal{R}_{\mathsf{CSC}}(s)$. We define the generator, encoder, prover, and verifier as follows.

$\underline{\mathcal{G}(1^\lambda, (m, N, \ell, t, q, d \in \mathbb{N})) \to \mathsf{pp}}$:

1. Let $n = 2 \cdot (\ell + 1)$
2. $\mathsf{pp}_{\mathsf{PC}} \leftarrow \mathsf{Gen}(1^\lambda, \log n - 1)$
3. Output $(m, n, N, \ell, t, q, d, \mathsf{pp}_{\mathsf{PC}})$

$\underline{\mathcal{K}(\mathsf{pp}, \mathsf{s}) \to (\mathsf{pk}, \mathsf{vk})}$:

1. Parse constants $(c_1, \ldots, c_q)$ from the structure s.
2. Let $\mathsf{pk} \leftarrow (\mathsf{pp}, \mathsf{s})$ and $\mathsf{vk} \leftarrow (\mathsf{pp}, (c_1, \ldots, c_q))$
3. Output $(\mathsf{pk}, \mathsf{vk})$

$\underline{\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (\mathsf{L}_{\{k \in [\mu]\}}, \mathsf{C}_{\{k \in [\nu]\}}))}$:

1. $\mathcal{V}, \mathcal{P}$: For size parameters m and n, let $s = \log m$ and $s' = \log n$. The prover parses the input structure s as

$$((\widetilde{M}_1, \ldots, \widetilde{M}_t), (S_1, \ldots, S_q), (c_1, \ldots, c_q)).$$

   The prover parses the input CCS instance-witness pairs L_k for $k \in [\mu]$ and linearized CCS instance-witness pairs C_k for $k \in [\nu]$ as

$$\begin{aligned} \Big((C_{1,k}, u_k, \mathsf{x}_{1,k}, r_k, v_{k,1}, \ldots, v_{k,t}), \widetilde{w}_{1,k}\Big) \quad & \text{for } k \in [\mu] \\ \Big((C_{2,k}, \mathsf{x}_{2,k}), \widetilde{w}_{2,k}\Big) \quad & \text{for } k \in [\nu]. \end{aligned}$$

   Let $\widetilde{z}_{1,k}$ be the multilinear extension of $(w_{1,k}, u_k, \mathsf{x}_{1,k})$ and let $\widetilde{z}_{2,k}$ be the multilinear extension of $(w_{2,k}, 1, \mathsf{x}_{2,k})$.

2. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\gamma \xleftarrow{\$} \mathbb{F}, \beta \xleftarrow{\$} \mathbb{F}^s$, and sends them to $\mathcal{P}$.

3. $\mathcal{P}$: Output a committed sumcheck witness polynomial g, represented as

$$((\widetilde{w}_{1,k})_{k\in[\mu]}, (\widetilde{w}_{2,k})_{k\in[\nu]}),$$

where

$$g(x) = \left(\sum_{j\in[t],k\in[\mu]} \gamma^{(k-1)\cdot t+j} \cdot L_{k,j}(x)\right) + \left(\sum_{k\in[\nu]} \gamma^{\mu\cdot t+k} \cdot Q_k(x)\right)$$

$$L_{k,j}(x) = \widetilde{eq}(r_k, x) \cdot \left(\sum_{y\in\{0,1\}^{s'}} \widetilde{M}_j(x,y) \cdot \widetilde{z}_{1,k}(y)\right)$$

$$Q_k(x) = \widetilde{eq}(\beta, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j\in S_i} \left(\sum_{y\in\{0,1\}^{s'}} \widetilde{M}_j(x,y) \cdot \widetilde{z}_{2,k}(y)\right)\right).$$

4. $\mathcal{P}, \mathcal{V}$: Output a committed sumcheck instance which consists of commitment to polynomial g

$$\overline{g} = ((C_{1,k}, u_k, \mathsf{x}_{1,k}, r_k)_{k\in[\mu]}, (C_{2,k}, \mathsf{x}_{2,k})_{k\in[\nu]}, \gamma, \beta),$$

claimed sum

$$T = \sum_{j\in[t],k\in[\mu]} \gamma^{(k-1)\cdot t+j} \cdot v_{k,j},$$

and an empty list of fixed evaluation points, reducing to the task of checking that the prover knows a polynomial g that corresponds with the commitment and that

$$T = \sum_{x\in\{0,1\}^s} g(x)$$

Lemma 4.7 (Folding CCS, Preprocessing). Construction 4.2 is a reduction of knowledge of type $\mathcal{R}^{\mu}_{\mathsf{LCCCS}} \times \mathcal{R}^{\nu}_{\mathsf{CCCS}} \to \mathcal{R}_{\mathsf{CSC}}(s)$.

Proof. Public reducibility follows by observation. Completeness and knowledge soundness follow from Lemma 4.8 and Lemma 4.9 respectively. □

Lemma 4.8 (Folding CCS, Preprocessing). Construction 4.2 is complete.

Proof. Consider the public parameters $\mathsf{pp} = (m, n, N, \ell, t, q, d, \mathsf{pp}_{\mathsf{PC}}) \leftarrow \mathcal{G}(1^\lambda, N)$ and let $s = \log m$ and $s' = \log n$. Consider arbitrary structure

$$\mathsf{s} = (\widetilde{M}_1, \ldots, \widetilde{M}_t), (S_1, \ldots, S_q), (c_1, \ldots, c_q) \leftarrow \mathcal{A}(\mathsf{pp}).$$

Consider the prover and verifier keys $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\mathsf{s}_1, \mathsf{s}_2))$. Suppose that the prover and the verifier are provided μ linearized committed CCS instances and ν committed CCS instances. Suppose that the prover additionally is provided with the corresponding satisfying witnesses.

Because the input linearized committed CCS instance-witness pairs are satisfying, we have for all $j \in [t]$ and $k \in [\mu]$

$$\begin{aligned}
v_{k,j} &= \sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(r_x, y) \cdot \widetilde{z}_{1,k}(y) && \text{By precondition.} \\
&= \sum_{x \in \{0,1\}^s} \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{1,k}(y) \right) && \text{By Lemma 4.5.} \\
&= \sum_{x \in \{0,1\}^s} L_{k,j}(x) && \text{By construction.}
\end{aligned}$$

Moreover, because the input committed CCS instance-witness pairs are satisfying, for all $k \in [\nu]$, we have, for all $x \in \{0,1\}^s$ that

$$0 = \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{2,k}(y) \right)$$

Treating the right-hand side of the above equation as a polynomial in x, because it vanishes on all $x \in \{0,1\}^s$, we have that it must be the zero polynomial. Therefore, we have, for β sampled by the verifier, that for all $k \in [\nu]$

$$\begin{aligned}
0 &= \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(\beta, y) \cdot \widetilde{z}_{2,k}(y) \right) \\
&= \sum_{x \in \{0,1\}^s} \widetilde{eq}(\beta, x) \cdot \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{2,k}(y) \right) && \text{By Lemma 4.5.} \\
&= \sum_{x \in \{0,1\}^s} Q_k(x) && \text{By construction.}
\end{aligned}$$

Therefore, for γ sampled by the verifier, by linearity, we have that

$$\begin{aligned}
T &= \sum_{j \in [t], k \in [\mu]} \gamma^{(k-1) \cdot t + j} \cdot v_{k,j} \\
&= \sum_{x \in \{0,1\}^s} \left(\left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1) \cdot t + j} \cdot L_{k,j}(x) \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot Q_k(x) \right) \right) \\
&= \sum_{x \in \{0,1\}^s} g(x)
\end{aligned}$$

□

Lemma 4.9 (Folding CCS, Preprocessing). Construction 4.2 is knowledge sound.

Proof. We prove knowledge soundness by proving tree extractability (Lemma 2.2). Indeed, suppose an extractor $\mathcal{X}$ is provided with structure

$$\mathsf{s} = ((\widetilde{M}_1, \ldots, \widetilde{M}_t), (S_1, \ldots, S_q), (c_1, \ldots, c_q)),$$

input $\mathcal{R}_{\mathsf{LCCCS}}$ instances

$$(C_{1,k}, u_k, \mathsf{x}_{1,k}, r_k, v_{k,1}, \ldots, v_{k,t}) \quad \text{for } k \in [\mu]$$

and input $\mathcal{R}_{\mathsf{CCCS}}$ instances

$$(C_{2,k}, \mathsf{x}_{2,k}) \quad \text{for } k \in [\nu].$$

Moreover, suppose $\mathcal{X}$ is provided $m \cdot (\mu \cdot t + \nu)$ accepting transcripts with corresponding output witnesses $((\widetilde{w}_{1,k})_{k\in[\mu]}, (\widetilde{w}_{2,k})_{k\in[\nu]})$ varying over randomness $\beta^{(i')}$ for $i' \in [m]$ and for each $\beta^{(i')} \in [m]$ varying randomness $\gamma^{(i',i)}$ for $i \in [\mu \cdot t + \nu]$. Note that the output satisfying witnesses are identical across transcripts because all corresponding output commitments are identical. We argue that these output witnesses are precisely satisfying input witnesses.

Indeed, by the satisfiability of the output witnesses, we have that

$$\begin{aligned}
T^{(i',i)} &= \sum_{j\in[t],k\in[\mu]} {\gamma^{(i)}}^{(k-1)\cdot t+j} \cdot v_{k,j} + \sum_{k\in[\nu]} {\gamma^{(i',i)}}^{\mu\cdot t+k} \cdot 0 \\
&= \sum_{x\in\{0,1\}^s} g^{(i',i)}(x) \\
&= \sum_{x\in\{0,1\}^s} \left(\left(\sum_{j\in[t],k\in[\mu]} {\gamma^{(i)}}^{(k-1)\cdot t+j} \cdot L_{k,j}(x) \right) + \left(\sum_{k\in[\nu]} {\gamma^{(i',i)}}^{\mu\cdot t+k} \cdot Q_k^{(i')}(x) \right) \right) \\
&= \sum_{j\in[t],k\in[\mu]} {\gamma^{(i',i)}}^{(k-1)\cdot t+j} \cdot \left(\sum_{x\in\{0,1\}^s} L_{k,j}(x) \right) + \sum_{k\in[\nu]} {\gamma^{(i',i)}}^{\mu\cdot t+k} \cdot \left(\sum_{x\in\{0,1\}^s} Q_k^{(i')}(x) \right)
\end{aligned}$$

Note that $Q_k^{(i')}$ remains identical for all $\gamma^{(i',i)}$ for all $i \in [\mu\cdot t+\nu]$ due to the corresponding commitments remaining identical. Interpolating over the choice of γ, we have for all $j \in [t]$ and $k \in [\mu]$

$$v_{k,j} = \sum_{x\in\{0,1\}^s} L_{k,j}(x),$$

and for all $k \in [\nu]$ and $i' \in [m]$

$$0 = \sum_{x\in\{0,1\}^s} Q_k^{(i')}(x).$$

Then, for all $j \in [t]$ and $k \in [\mu]$, we have

$$v_{k,j} = \sum_{x\in\{0,1\}^s} L_{k,j}(x)$$

$$= \sum_{x \in \{0,1\}^s} \widetilde{eq}(r_x, x) \cdot \left(\sum_{y \in \{0,1\}^{s'}} M_j(x, y) \cdot \widetilde{z}_{1,k}(y) \right)$$
$$= \sum_{y \in \{0,1\}^{s'}} M_j(r_x, y) \cdot \widetilde{z}_{1,k}(y) \qquad \text{By Lemma 4.5}$$

This implies that $(\widetilde{w}_{1,k})_{k \in [\mu]}$ are satisfying witnesses for the $\mathcal{R}_{\mathsf{LCCCS}}$ instances.

Finally, we have that for all $k \in [\nu]$ and $i' \in [m]$

$$0 = \sum_{x \in \{0,1\}^s} Q_k^{(i')}(x)$$
$$= \sum_{x \in \{0,1\}^s} \widetilde{eq}(\beta^{(i')}, x) \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{2,k}(y) \right) \right)$$
$$= \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(\beta^{(i')}, y) \cdot \widetilde{z}_{2,k}(y) \right)$$

By interpolating over the choice of β, we have that for all $x \in \{0,1\}^s$

$$0 = \sum_{i=1}^{q} c_i \cdot \prod_{j \in S_i} \left(\sum_{y \in \{0,1\}^{s'}} \widetilde{M}_j(x, y) \cdot \widetilde{z}_{2,k}(y) \right)$$

This implies that $(\widetilde{w}_{2,k})_{k \in [\nu]}$ are satisfying witnesses for the $\mathcal{R}_{\mathsf{CCCS}}$ instances. □

We now describe the sumcheck protocol, recast as a reduction of knowledge from the committed sumcheck relation to the polynomial evaluation relation. We begin by describing the central recursive reduction, which reduces the task of checking the a sumcheck instance over m variables to the task of checking a sumcheck instance over $m - 1$ variables.

Construction 4.3 (Committed Sumcheck Reduction [106]). We construct a reduction of knowledge of type $\mathcal{R}_{\mathsf{CSC}}(m) \to \mathcal{R}_{\mathsf{CSC}}(m-1)$. Our construction is polymorphic over all generator and encoder algorithms and all underlying commitment schemes.

$\underline{\langle \mathcal{P}, \mathcal{V} \rangle((\overline{g}, T, (r_{m+1}, \ldots, r_n)), g)}$:

1. $\mathcal{P}$: Send to $\mathcal{V}$ the polynomial

$$g_m(X_m) = \sum_{x_1, \ldots, x_{m-1} \in \{0,1\}} g(x_1, \ldots, x_{m-1}, X_m, r_{m+1}, \ldots, r_n)$$

2. $\mathcal{V}$: Check that

$$T = \sum_{x_m \in \{0,1\}} g_m(x_m)$$

and send to $\mathcal{P}$ random challenge $r_m \in \mathbb{F}$.

3. $\mathcal{P}, \mathcal{V}$: Compute $T' \leftarrow g_m(r_m)$ and output the updated instance

$$(\overline{g}, T', (r_m, \ldots, r_n))$$

4. $\mathcal{P}$: Output the input witness g as the updated witness.

Lemma 4.10 (Committed Sumcheck Reduction [106]). Construction 4.3 is a reduction of knowledge of type $\mathcal{R}_{\mathsf{CSC}}(m) \to \mathcal{R}_{\mathsf{CSC}}(m-1)$.

Corollary 4.1 (Sumcheck Protocol). Consider the polynomial evaluation relation, $\mathcal{R}_{\mathsf{polyeval}}$, as defined in Definition 4.10. Let Π_{CSC} be the reduction of knowledge in Construction 4.3. Then

$$\underbrace{\Pi_{\mathsf{CSC}} \circ \cdots \circ \Pi_{\mathsf{CSC}}}_{m \text{ times}}$$

is a reduction of knowledge of type $\mathcal{R}_{\mathsf{CSC}}(m) \to \mathcal{R}_{\mathsf{polyeval}}$.

Construction 4.4 (Folding CCS, Postprocessing). We construct a reduction of knowledge of type $\mathcal{R}_{\mathsf{polyeval}} \to \mathcal{R}_{\mathsf{LCCCS}}$. We define the generator and encoder identically as Construction 4.2. We define the prover, and verifier as follows.

$\underline{\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \overline{g}, r'_x, c) \to \mathsf{L}}$:

1. $\mathcal{P}, \mathcal{V}$: The prover parses the commitment $\overline{g}$ as

$$((C_{1,k}, u_k, \mathsf{x}_{1,k}, r_k)_{k\in[\mu]}, (C_{2,k}, \mathsf{x}_{2,k})_{k\in[\nu]}, \gamma, \beta)$$

and parses

$$((\widetilde{M}_1, \ldots, \widetilde{M}_t), (S_1, \ldots, S_q), (c_1, \ldots, c_q)).$$

from the prover key. The verifier parses $(c_1, \ldots, c_q)$ from the verifier key. Let $\widetilde{z}_{1,k}$ be the multilinear extension of $(w_{1,k}, u_k, \mathsf{x}_{1,k})$ and let $\widetilde{z}_{2,k}$ be the multilinear extension of $(w_{2,k}, 1, \mathsf{x}_{2,k})$.

2. $\mathcal{P} \to \mathcal{V}$: Compute and send

$$\sigma_{k,j} \leftarrow \sum_{y\in\{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z}_{1,k}(y) \quad \text{for all } k \in [\mu], j \in [t],$$
$$\theta_{k,j} \leftarrow \sum_{y\in\{0,1\}^{s'}} \widetilde{M}_j(r'_x, y) \cdot \widetilde{z}_{2,k}(y) \quad \text{for all } k \in [\nu], j \in [t]$$

3. $\mathcal{V}$: Compute $e_{1,k} \leftarrow \widetilde{eq}(r_k, r'_x)$ and $e_2 \leftarrow \widetilde{eq}(\beta, r'_x)$, and check that

$$c = \left(\sum_{j\in[t], k\in[\mu]} \gamma^{(k-1)\cdot t + j} \cdot e_{1,k} \cdot \sigma_{j,k} \right) + \left(\sum_{k\in[\nu]} \gamma^{\mu\cdot t + k} \cdot e_2 \cdot \left(\sum_{i=1}^{q} c_i \cdot \prod_{j\in S_i} \theta_{j,k} \right) \right)$$

4. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\rho \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$.

5. $\mathcal{V}, \mathcal{P}$: Output the folded linearized committed CCS instance $(C, u, \mathsf{x}, r'_x, v_1, \ldots, v_t)$, where for all $j \in [t]$:

$$\begin{aligned} C &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot C_{1,k} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot C_{2,k} \\ u &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot u_k + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot 1 \\ \mathsf{x} &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot \mathsf{x}_{1,k} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \mathsf{x}_{2,k} \\ v_j &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot \sigma_{j,k} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \theta_{j,k} \end{aligned}$$

6. $\mathcal{P}$: Output the folded witness $\widetilde{w} \leftarrow \sum_{k\in[\mu]} \rho^k \cdot \widetilde{w}_{1,k} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \widetilde{w}_{2,k}$.

Lemma 4.11. Construction 4.4 is a reduction of knowledge from $\mathcal{R}_{\mathsf{polyeval}}$ (with respect to the described commitment scheme) to $\mathcal{R}_{\mathsf{LCCCS}}$.

Proof. Public reducibility follows from observation. Completeness and knowledge soundness follow from Lemma 4.12 and Lemma 4.13 respectively. □

Lemma 4.12 (Folding CCS, Postprocessing). Construction 4.4 is complete.

Proof. By the precondition, we have that $c = g(r'_x)$. Therefore, we have for $e_{1,k} = \widetilde{eq}(r_k, r'_x)$ for $k \in [\mu]$, $e_2 = \widetilde{eq}(\beta, r'_x)$,

$$\sigma_{k,j} = \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z}_{1,k}(y),$$

for $j \in [t]$ and $k \in [\mu]$, and

$$\theta_{k,j} = \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z}_{2,k}(y)$$

$j \in [t]$ and $k \in [\nu]$ that

$$\begin{aligned} c &= g(r'_x) \\ &= \left(\left(\sum_{j\in[t],k\in[\mu]} \gamma^{(k-1)\cdot t+j} \cdot L_{j,k}(r'_x) \right) + \left(\sum_{k\in[\nu]} \gamma^{\mu\cdot t+k} \cdot Q_k(r'_x) \right) \right) \\ &= \left(\left(\sum_{j\in[t],k\in[\mu]} \gamma^{(k-1)\cdot t+j} \cdot e_1 \cdot \sigma_{j,k} \right) + \left(\sum_{k\in[\nu]} \gamma^{\mu\cdot t+k} \cdot e_2 \sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \theta_{j,k} \right) \right). \end{aligned}$$

This implies that the verifier's intermediate check passes.

Now, consider the following linearized committed CCS instances obtained by reducing input committed CCS instances (for all $k \in [\nu]$):

$$(C_{2,k}, 1, \mathsf{x}_{2,k}, r'_x, \theta_{k,1}, \ldots, \theta_{k,t}).$$

By the precondition that g is a satisfying opening to $\overline{g}$ and by the definition of $(\theta_{1,k}, \ldots, \theta_{t,k})$ for all $k \in [\nu]$, we have that kth linearized committed CCS instance is satisfied by the witness of the kth committed CCS instances i.e., $\widetilde{w}_{2,k}$.

Therefore, for a random ρ sampled by the verifier, and for

$$\begin{aligned} C &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot C_{1,k} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot C_{2,k} \\ u &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot u_k + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot 1 \\ \mathsf{x} &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot \mathsf{x}_{1,k} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \mathsf{x}_{2,k} \\ v_j &\leftarrow \textstyle\sum_{k\in[\mu]} \rho^k \cdot \sigma_{k,j} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \theta_{k,j} \end{aligned}$$

we have that the output linearized CCS instance

$$(C, u, \mathsf{x}, r'_x, v_1, \ldots, v_t)$$

is satisfied by the witness

$$\widetilde{w} \leftarrow \sum_{k\in[\mu]} \rho^k \cdot \widetilde{w}_{1,k} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \widetilde{w}_{2,k}$$

by linearity and the additive homomorphism property of the polynomial commitment scheme. □

Lemma 4.13 (Folding CCS, Postprocessing). Construction 4.4 is knowledge sound.

Proof. We prove knowledge soundness by proving tree extractability (Lemma 2.2). Indeed, suppose an extractor $\mathcal{X}$ is provided with input structure

$$\mathsf{s} = ((\widetilde{M}_1, \ldots, \widetilde{M}_t), (S_1, \ldots, S_q), (c_1, \ldots, c_q)).$$

and input instance $(\overline{g}, r'_x, c)$, where the commitment $\overline{g}$ is parsed as

$$((C_{1,k}, u_k, \mathsf{x}_{1,k}, r_k)_{k\in[\mu]}, (C_{2,k}, \mathsf{x}_{2,k})_{k\in[\nu]}, \gamma, \beta).$$

Moreover, suppose $\mathcal{X}$ is provided with a tree of accepting transcripts consisting of satisfying output $\mathcal{R}_{\mathsf{LCCCS}}$ instance-witness pairs $(\mathsf{u}^{(i)}, \mathsf{w}^{(i)})$ and corresponding final random challenge $\rho^{(i)}$ for $i \in \{1, \ldots, \mu + \nu\}$. The extractor begins by interpolating points $(\rho^{(i)}, \mathsf{w}^{(i)})$ for all $i \in [\mu+\nu]$ to retrieve μ witnesses $\mathsf{w}_1 = (\widetilde{w}_{1,1}, \ldots, \widetilde{w}_{1,\mu})$ and ν witnesses $\mathsf{w}_2 = (\widetilde{w}_{2,1}, \ldots, \widetilde{w}_{2,\nu})$ such that for $i \in [\mu + \nu]$

$$\mathsf{w}^{(i)} = \sum_{k\in[\mu]} \rho^k \cdot \widetilde{w}_{1,k} + \sum_{k\in[\nu]} \rho^{\mu+k} \cdot \widetilde{w}_{2,k}. \tag{4.21}$$

The extractor then produces the input witness $(\mathsf{w}_1, \mathsf{w}_2)$ representing the witness portion of the polynomial g. We will show that the witness produced by $\mathcal{X}$ is a satisfying witness polynomial for the input instance.

Indeed, let $\mathsf{u}^{(i)} = (C^{(i)}, u^{(i)}, \mathsf{x}^{(i)}, r_x^{(i)}, v_1^{(i)}, \ldots, v_t^{(i)})$. We first show that the retrieved polynomials are valid openings to the corresponding commitments in the instance. For $i \in \{1, \ldots, \mu + \nu\}$, because $\mathsf{w}^{(i)}$ is a satisfying witness, by construction,

$$\begin{aligned}
&\sum_{k\in[\mu]} {\rho^{(i)}}^k \cdot \mathsf{Commit}(\mathsf{pp}, \widetilde{w}_{1,k}) + \sum_{k\in[\nu]} {\rho^{(i)}}^{\mu+k} \cdot \mathsf{Commit}(\mathsf{pp}, \widetilde{w}_{2,k}) \\
&= \mathsf{Commit}\left(\mathsf{pp}, \left(\sum_{k\in[\mu]} {\rho^{(i)}}^k \cdot \widetilde{w}_{1,k}\right) + \left(\sum_{k\in[\nu]} {\rho^{(i)}}^{\mu+k} \cdot \widetilde{w}_{2,k}\right)\right) && \text{By additive homomorphism.} \\
&= \mathsf{Commit}(\mathsf{pp}, \mathsf{w}^{(i)}) && \text{By Equation (4.21).} \\
&= C^{(i)} && \text{Witness } \mathsf{w}^{(i)} \text{ is satisfying.} \\
&= \sum_{k\in[\mu]} {\rho^{(i)}}^k \cdot C_{1,k} + \sum_{k\in[\nu]} {\rho^{(i)}}^{\mu+k} \cdot C_{2,k} && \text{By verifier's computation.}
\end{aligned}$$

Interpolating, we have that for all $i \in [\mu]$ and $j \in [\nu]$

$$\mathsf{Commit}(\mathsf{pp}, \mathsf{w}_{1,i}) = C_{1,i} \tag{4.22}$$
$$\mathsf{Commit}(\mathsf{pp}, \mathsf{w}_{2,j}) = C_{2,j}. \tag{4.23}$$

Next, we must argue that w_1 and w_2 satisfy the remainder of the instances u_1 and u_2 respectively under the structure s. Indeed, consider $\{\sigma_{k,j}\}$ (for all $j \in [t]$ and $k \in [\mu]$), and $\{\theta_{k,j}\}$ (for all $j \in [t]$ and $k \in [\nu]$) sent by the prover which by the extractor's construction are identical across all executions of the interaction. By the verifier's computation we have that for $i \in \{1, \ldots, \mu + \nu\}$ and all $j \in [t]$

$$\sum_{k\in[\mu]} (\rho^{(i)})^k \cdot \sigma_{k,j} + \sum_{k\in[\nu]} (\rho^{(i)})^{\mu+k} \cdot \theta_{k,j} = v_j^{(i)} \tag{4.24}$$

Now, because $\mathsf{w}^{(i)}$ is a satisfying witness, for $i \in \{1, \ldots, \mu + \nu\}$ we have for all $j \in [t]$ that

$$v_j^{(i)} = \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z}^{(i)}(y),$$

where $\widetilde{z}^{(i)} = (\widetilde{w^{(i)}, u^{(i)}}, \mathsf{x}^{(i)})$ where $w^{(i)}$ is the result of interpreting $\mathsf{w}^{(i)}$ as a multilinear polynomial.

However, by Equations (4.21) and (4.24), for $i \in \{1, \ldots, \mu + \nu\}$ and $j \in [t]$, this implies that

$$\begin{aligned}
&\sum_{k\in[\mu]} (\rho^{(i)})^k \cdot \sigma_{k,j} + \sum_{k\in[\nu]} (\rho^{(i)})^{\mu+k} \cdot \theta_{k,j} \\
&= \sum_{k\in[\mu]} (\rho^{(i)})^k \cdot \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{1,k}}(y) + \sum_{k\in[\nu]} (\rho^{(i)})^{\mu+k} \cdot \sum_{y\in\{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{2,k}}(y),
\end{aligned}$$

where $\widetilde{z_{1,k}} = (\widetilde{w_{1,k}, u_k}, \mathsf{x}_{1,k})$ for $k \in [\mu]$ where $w_{1,k}$ denotes the multilinear polynomial interpretation of $\mathsf{w}_{1,k}$ and $\widetilde{z_{2,k}} = (\widetilde{w_{2,k}, 1, \mathsf{u}_{2,k}.\mathsf{x}})$ for $k \in [\nu]$ where $w_{2,k}$ represents the multilinear polynomial interpretation of $\mathsf{w}_{2,k}$. Interpolating, we have that, for all $j \in [t]$

$$\sigma_{k,j} = \sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{1,k}}(y) \quad \text{for all } k \in [\mu]$$
$$\theta_{k,j} = \sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{2,k}}(y) \quad \text{for all } k \in [\nu].$$

Thus, because that the verifier does not abort, we have that

$$\begin{aligned}
c &= \left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot e_1 \cdot \sigma_{j,k} \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot e_2 \cdot \sum_{i \in [q]} c_i \cdot \prod_{j \in S_i} \theta_j \right) \\
&= \left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot \widetilde{eq}(r_x, r'_x) \cdot \sigma_{j,k} \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot \widetilde{eq}(\beta, r'_x) \cdot \sum_{i \in [q]} c_i \cdot \prod_{j \in S_i} \theta_{j,k} \right) \\
&= \left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot \widetilde{eq}(r_x, r'_x) \cdot \sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{1,k}}(y) \right) + \\
&\quad \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot \widetilde{eq}(\beta, r'_x) \cdot \sum_{i \in [q]} c_i \cdot \prod_{j \in S_i} \sum_{y \in \{0,1\}^{s'}} \widetilde{M_j}(r'_x, y) \cdot \widetilde{z_{2,k}}(y) \right) \\
&= \left(\left(\sum_{j \in [t], k \in [\mu]} \gamma^{(k-1)\cdot t + j} \cdot L_{j,k}(r'_x) \right) + \left(\sum_{k \in [\nu]} \gamma^{\mu \cdot t + k} \cdot Q_k(r'_x) \right) \right) \\
&= g(r'_x)
\end{aligned}$$

This implies that the extractor $\mathcal{X}$ has indeed extracted a satisfying witness. □

Putting everything together we have the main result of this section.

Theorem 4.2 (Folding CCS). Let Π_{pre} be the reduction of knowledge of type $\mathcal{R}^{\mu}_{\mathsf{LCCCS}} \times \mathcal{R}^{\nu}_{\mathsf{CCCS}} \to \mathcal{R}_{\mathsf{CSC}}(s)$ from Construction 4.2. Let Π_{SC} be the reduction of knowledge of type $\mathcal{R}_{\mathsf{CSC}}(s) \to \mathcal{R}_{\mathsf{polyeval}}$ from Corollary 4.1. Let Π_{post} be the reduction of knowledge of type $\mathcal{R}_{\mathsf{polyeval}} \to \mathcal{R}_{\mathsf{LCCCS}}$ from Construction 4.4. Then

$$\Pi_{\mathsf{post}} \circ \Pi_{\mathsf{SC}} \circ \Pi_{\mathsf{pre}}$$

is a public-coin, succinct reduction of knowledge of type $\mathcal{R}^{\mu}_{\mathsf{LCCCS}} \times \mathcal{R}^{\nu}_{\mathsf{CCCS}} \to \mathcal{R}_{\mathsf{LCCCS}}$.

Chapter 5

Recursion from Folding

This chapter contains joint work with Srinath Setty and Ioanna Tzialla [99, 101].

The main idea is to construct recursively embedded proofs: to merge proofs π_1 and π_2, I prove that "I have seen convincing π_1 and π_2".

– Paul Valiant,
Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency

5.1 Incrementally Verifiable Computation

Recall that incrementally verifiable computation (IVC) is a special type of non-interactive proof of knowledge that demonstrates that n applications of a (non-deterministic) function F on some initial input z_0 results in output z_n. More formally, IVC concerns computational instances of the form (F, n, z_0, z_n). A satisfying witness to this instance is the list of non-deterministic inputs $(\omega_0, \ldots, \omega_{n-1})$ such that for $z'_0 = z_0$ and $z'_{i+1} \leftarrow F(z'_i, \omega_i)$, we have that $z'_n = z_n$. A key requirement is that a proof Π_i of i steps can be efficiently updated (in other words, incremented) to produce a proof Π_{i+1} of $i+1$ steps that does not grow in

size.

To achieve such a construction, recall that Valiant [133] relies on proof recursion, in which a proof of knowledge for $i+1$ applications attests to a single application of F, as well as the existence of a valid proof of knowledge for i applications (Figure 1.1). Unfortunately, this requires having to represent a SNARK verifier inside an arithmetic circuit, which is prohibitively expensive in practice as discussed in Section 1.5.

In this section, we develop a highly efficient incrementally verifiable computation scheme by utilizing reductions of knowledge, namely folding schemes, as opposed to proofs of knowledge. Looking forward, in Section 5.1.1 we begin by defining IVC, both traditionally and in the reductions of knowledge framework. In Section 5.1.2, we provide an informal overview of our IVC construction before providing a formal construction and proofs of correctness in Section 5.1.4. A distinctive aspect of our approach to IVC is that we achieve the smallest "verifier circuit" in the literature. We compare the concrete costs of our approach to the extant literature in Section 5.1.5.

5.1.1 Defining IVC

We begin by recalling the traditional formulation of incrementally verifiable computation, as defined by Valiant [133]. Informally, completeness holds if given an accepting proof Π_i for a statement (i, z_0, z_i) and a witness ω_i such that $z_{i+1} = F(z_i, \omega_i)$, the prover is guaranteed to produce an accepting proof Π_{i+1} for statement $(i+1, z_0, z_{i+1})$. Similarly, knowledge soundness holds if for any malicious prover $\mathcal{P}^*$ that is able to produce an accepting proof Π_i for statement (i, z_0, z_i), there exists a corresponding extractor $\mathcal{E}$ that can produce the corresponding witnesses $(\omega_0, \ldots, \omega_{i-1})$.

Definition 5.1 (Incrementally Verifiable Computation (IVC), Traditional). An incrementally verifiable computation (IVC) scheme is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic $\mathcal{K}$ denoting the generator, the prover, the verifier, and the encoder respectively, with the following interface

- $\mathcal{G}(1^\lambda, N) \to \mathsf{pp}$: on input security parameter λ and size bounds N, samples public parameters pp.
- $\mathcal{K}(\mathsf{pp}, F) \to (\mathsf{pk}, \mathsf{vk})$: on input public parameters pp, and polynomial-time function F, deterministically produces a prover key pk and a verifier key vk.
- $\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \to \Pi_{i+1}$: on input a prover key pk, a counter i, an initial input z_0, a claimed output after i iterations z_i, a non-deterministic advice ω_i, and an IVC proof Π_i attesting to z_i, produces a new proof Π_{i+1} attesting to $z_{i+1} = F(z_i, \omega_i)$.
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \to \{0, 1\}$: on input a verifier key vk, a counter i, an initial input z_0, a claimed output after i iterations z_i, and an IVC proof Π_i attesting to z_i, outputs 1 if Π_i is accepting, and 0 otherwise.

An IVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies the following requirements.

1. Perfect Completeness: For any PPT adversary $\mathcal{A}$

$$\Pr\left[\mathcal{V}(\mathsf{vk},(i+1,z_0,z_{i+1}),\Pi_{i+1})=1 \;\middle|\; \begin{array}{l}\mathsf{pp}\leftarrow\mathcal{G}(1^\lambda,N),\\ F,(i,z_0,z_i,\Pi_i)\leftarrow\mathcal{A}(\mathsf{pp}),\\ (\mathsf{pk},\mathsf{vk})\leftarrow\mathcal{K}(\mathsf{pp},F),\\ z_{i+1}\leftarrow F(z_i,\omega_i),\\ \mathcal{V}(\mathsf{vk},i,z_0,z_i,\Pi_i)=1,\\ \Pi_{i+1}\leftarrow\mathcal{P}(\mathsf{pk},(i,z_0,z_i),\omega_i,\Pi_i)\end{array}\right]=1$$

   where F is a polynomial-time computable function represented as an arithmetic circuit.

2. Knowledge Soundness: Consider constant $n\in\mathbb{N}$. For all expected polynomial-time adversaries $\mathcal{P}^*$ there exists an expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr_{\mathsf{r}}\left[\begin{array}{l}z_n=z \text{ where}\\ z_{i+1}\leftarrow F(z_i,\omega_i)\\ \forall i\in\{0,\ldots,n-1\}\end{array}\;\middle|\; \begin{array}{l}\mathsf{pp}\leftarrow\mathcal{G}(1^\lambda,N),\\ (F,(z_0,z_i),\Pi)\leftarrow\mathcal{P}^*(\mathsf{pp},\mathsf{r}),\\ (\mathsf{pk},\mathsf{vk})\leftarrow\mathcal{K}(\mathsf{pp},F),\\ \mathcal{V}(\mathsf{vk},(n,z_0,z),\Pi)=1,\\ (\omega_0,\ldots,\omega_{n-1})\leftarrow\mathcal{E}(\mathsf{pp},\mathsf{r})\end{array}\right]\approx 1$$

   where r denotes an arbitrarily long random tape. Moreover, F is a polynomial-time computable function represented as an arithmetic circuit.

3. Succinctness: The size of an IVC proof Π is independent of the number of iterations n.

While this definition can be intuitively understood, it has a key incongruence: While completeness is defined with respect to a single step (i.e., given a valid proof Π_i, the prover can produce a valid proof Π_{i+1}), knowledge soundness is defined globally (i.e., given a valid proof Π_i the extractor can pull out all prior witnesses from the beginning of time). This incongruence is far from cosmetic. As all known IVC constructions utilize recursion, a global extractor-based definition becomes problematic in this regime: Recursive proofs require *recursive extraction* in which the extractor for step $n-1$ plays the malicious prover for the extractor at step n. This incurs a polynomial blowup in the extractor for each successive recursive step. In particular, this results in a final extractor that runs in exponential-time with respect to the recursion-depth, which disqualifies it as a valid extractor. Indeed, Hall-Andersen and Nielsen [87] indicate that polynomial-depth recursion may be infeasible in well-studied models, such as the random oracle model, where every party is given access to idealized (and reproducible) randomness.

As such, the traditional definition of IVC seems to necessarily bake in non-standard assumptions. We circumvent this issue by utilizing reductions of knowledge to capture the security requirement for a single step of IVC. This enables us to disentangle the core IVC soundness definition from the additional assumptions needed to get polynomial-depth IVC. Informally, we define incrementally verifiable computation as a reduction of knowledge from the task of checking a proof of knowledge of i steps and and the latest evaluation of F to the task of checking a proof of knowledge of $i+1$ steps (of the same size).

Crucially, this does not contradict the logarithmic-depth barrier, as internally the proof of Theorem 1.1 requires having the extractor of Π_2 play the successful prover for the extractor of Π_1. As such, reductions of knowledge can also only be composed a logarithmic number of times to avoid an exponential blowup in the extractor time complexity. The key affordance of our definition then is a clean setting to study IVC without burdening ourselves with the additional complications of polynomial-depth recursion.

To formally define IVC in the reductions of knowledge framework, we first recall the Eval relation, which captures statements of the form *"I know w such that $y = F(x, w)$"*.

Definition 5.2 (Evaluation Relation). We define the evaluation relation Eval as follows

$$\mathsf{Eval} = \left\{ F, (x, y), w \;\middle|\; \begin{array}{l} x, y \in \mathsf{U}, w \in \mathsf{W}, \\ y = F(x, w) \end{array} \right\}$$

where $F : \mathsf{U} \times \mathsf{W} \to \mathsf{U}$ is an arithmetic circuit.

We now formally define IVC using refined reductions of knowledge (Definition 2.13).

Definition 5.3 (Incrementally Verifiable Computation (IVC)). An incrementally verifiable computation scheme is defined by (Proof, Π) where

(i) Proof is a ternary relation over a structure consisting of arithmetic circuit $F : \mathsf{U} \times \mathsf{W} \to \mathsf{U}$, an instance consisting of (i, z_0, z_i) where $i \in \mathbb{N}$ represents the iteration count, $z_0 \in \mathsf{U}$ represents the initial input, and $z_i \in \mathsf{U}$ represents the final output, and a fixed-size witness (referred to as the IVC proof).

(ii) Π is a zero-interactive reduction of knowledge of type $\mathsf{Eval} \overset{\sim_1}{\times} \mathsf{Proof} \overset{\sim_2}{\longrightarrow} \mathsf{Proof}$ where $(z_i, z_{i+1}) \sim_1 (i, z_0, z_i)$ and $((z_i, z_{i+1}), (i, z_0, z_i)) \sim_2 (i + 1, z_0, z_{i+1})$.

5.1.2 Overview

We begin by overviewing our construction for IVC from folding schemes. Consider an arithmetic circuit F represented as, say, R1CS constraints. Recall that the witness for an IVC statement $(i + 1, z_0, z_{i+1})$ is **(1)** an IVC proof Π_i attesting knowledge of a witness for an IVC statement (i, z_0, z_i), and **(2)** an input ω_i such that $z_{i+1} = F(z_i, \omega_i)$.

We now describe a single iterative step of the prover's work. That is, we explain how the prover can take a proof Π_i for the IVC statement (i, z_0, z_i) alongside a new input ω_i and efficiently produce an updated proof Π_{i+1} for the IVC statement $(i + 1, z_0, z_{i+1})$. At a high level, instead of directly proving the correct execution of F in each step, the prover proves the correct execution of an augmented function F'. The augmented function F', in addition to running F, performs additional bookkeeping using a folding scheme to help verifiably update the IVC proof.

In particular, on top of running F, F' also takes as input an R1CS instance u_i that claims the correct execution of iteration i of F' and a relaxed R1CS instance U_i that claims the correct execution of all prior iterations of F'. Instead of directly checking these instance (which would be concretely expensive), F' folds u_i into instance in U_i to produce

an updated relaxed R1CS instance U_{i+1}. To claim the correctness of F' itself, the prover produces a new instance u_{i+1}. Now, by the soundness of the folding scheme, checking U_{i+1} is equivalent to checking all i iterations of F'. Thus, checking both u_{i+1} and U_{i+1} is equivalent to checking $i+1$ iterations of F', completing the recursion cycle.

As such, we let the IVC proof Π_i contain the running instance U_i, the fresh instance u_i, and the corresponding witnesses. Then, the prover can use parts of Π_i as input to F' to produce U_{i+1} and u_{i+1}, and separately compute the corresponding witnesses. These terms together define Π_{i+1}. We now provide additional details. We intentionally overlook certain minor complications, which we address before providing a formal construction.

The augmented function The function F' takes as private input the statement so far (i, z_0, z_i), the auxiliary witness ω_i, an R1CS instance u_i that claims that the step i was executed correctly, and a running relaxed R1CS instance U_i that attests to all prior iterations of F'. Function F' first runs F on input (z_i, ω_i) to compute z_{i+1}. As additional bookkeeping, F' runs a verifier circuit that does the following.

1. Checks that U_i is contained in the public output of the instance u_i (recall that this is the vector x in the R1CS instance). This enforces that U_i is indeed produced by the prior step.

2. Runs the non-interactive folding scheme's verifier to fold an instance that claims the correct execution of the previous step, u_i, into U_i to produce an updated running instance U_{i+1}. This ensures that checking U_{i+1} implies checking U_i and u_i while maintaining that U_{i+1} does not grow in size with respect to U_i.

F' produces as public output the new statement $(i+1, z_0, z_{i+1})$, and the updated list of running instances U_{i+1}.

Structure of an IVC proof We now discuss the structure of an IVC proof and how it can be checked. Consider an IVC statement (i, z_0, z_i). Let the corresponding IVC proof be Π_i, which consists of a running instance U_i, the corresponding witness W_i, an instance that claims the correctness of the latest iteration u_i, and the corresponding witness w_i.

Suppose we have the following: So long as $(\mathsf{u}_i, \mathsf{w}_i)$ is a satisfying instance-witness pair with respect to augmented function F' and contains U_i in the public output, we have that checking U_i implies checking all prior iterations. Thus, the verifier can check the IVC statement (i, z_0, z_i) by checking **(1)** the fresh instance-witness pair $(\mathsf{u}_i, \mathsf{w}_i)$ is satisfying, **(2)** the public IO (i.e, the vector x) of u_i contains U_i, and **(3)** the running instance-witness pair $(\mathsf{U}_i, \mathsf{W}_i)$ is satisfying.

Updating an IVC proof Given a proof Π_i of i steps, the prover can efficiently produce a proof Π_{i+1} of $i+1$ steps. The core invariant we maintain is as follows: If checking Π_i indeed attests to i steps, then we must have that Π_{i+1} attests to $i+1$ steps while maintaining that Π_{i+1} does not grow in size. Indeed, assume that checking $\Pi_i = ((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$ is sufficient to verify the IVC claim (i, z_0, z_i). Suppose the prover is provided as input proof Π_i, a claim (i, z_0, z_i), and an auxiliary witness ω_i.

The prover proceeds as follows: Using the non-interactive folding scheme, the prover first folds the instance-witness pair $(\mathsf{u}_i, \mathsf{w}_i)$, which attests to the correctness of the last step into $(\mathsf{U}_i, \mathsf{W}_i)$. Let $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})$ denote the updated running instance-witness pair. Now, by assumption, so long as u_i contains U_i, we have that that checking $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})$ is equivalent to checking Π_i while maintaining that $|(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})| = |(\mathsf{U}_i, \mathsf{W}_i)|$. To account for the next step of execution, the prover computes

$$((i+1, z_0, z_{i+1}), \mathsf{U}_{i+1}) \leftarrow F'(\mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i)$$

and computes the corresponding claim of correct execution u_{i+1} and corresponding witness w_{i+1}. Now, we have that checking u_{i+1} attests to the following.

1. F produces z_{i+1} on input (z_i, ω_i).
2. The public IO of u_i contains U_i, and therefore U_i indeed attests to i steps so long as u_i is valid.
3. U_{i+1} was computed by folding u_i into U_i and therefore checking U_{i+1} is equivalent to checking Π_i.

Therefore, so long as u_{i+1} is valid, we have that checking U_{i+1} attests to i steps. Moreover, because u_{i+1} attests to the correctness of the latest step, checking u_{i+1} is sufficient to attest to $i+1$ iterations. This means that checking $\Pi_{i+1} = ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}))$ is sufficient to check the IVC statement $(i+1, z_0, z_{i+1})$.

Fixing minor complications The prior description overlooks the following minor issues, which we now address.

First, we described how to update a proof Π_i to produce a proof Π_{i+1}. However, we did not define a base case proof Π_0 and how the prover, the verifier, and the function F' handle the base case. At a high level, we have F' populate U with a satisfying running instances in the base case.

Second, the non-interactive folding scheme's verifier run by F' needs additional advice generated by the non-interactive folding scheme's prover. To address this, the prover provides additional non-deterministic input to F'.

Finally, there is a subtle sizing issue in the above description: in each step, because U_{i+1} is produced as the public IO of F', it must be contained in the public IO of instance u_{i+1}. However, in the next iteration, because the public IO of u_{i+1} is folded into the public IO of U_{i+1}, this means that we are stuck trying to squeeze U_{i+1} into the public IO of U_{i+1}. To alleviate this issue, we have each F' only produce a succinct commitment of its outputs as public output. In the subsequent step, F' takes as non-deterministic input an opening to this commitment.

5.1.3 IVC-Compatible Folding Schemes

Generalizing the above discussion, a succinct, non-interactive folding scheme for an arbitrary committed relation $\mathcal{R}_2$ can be used for IVC if **(1)** statements about the correct

execution of an efficient function F can be encoded (and decoded) as statements in the underlying relation of $\mathcal{R}_2$ in a way that preserves the size of F, **(2)** structures and instances can be encoded (and decoded) independently of witnesses, and **(3)** there exists a default satisfying instance-witness pair in $\mathcal{R}_2$. We formally define IVC-compatibility as follows.

Definition 5.4 (IVC-Compatible Folding Scheme). Consider a relation $\mathcal{R}_1$, and a committed relation $\mathcal{R}_2$ over an underlying relation $\mathcal{R}'_2$. A succinct, non-interactive folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ with deterministic $\mathcal{V}$ of type $\mathcal{R}_1^{\mu} \times \mathcal{R}_2^{\nu} \to \mathcal{R}_1$ is IVC-compatible if it satisfies the following properties.

1. NP-completeness: There exists a deterministic polynomial-time efficiently invertible function enc, such that for any arithmetic circuit F, $\mathsf{enc}(F, (x, y), w) \in \mathcal{R}'_2$ if and only if $F(x, w) = y$.
2. Partial functions: There exists deterministic polynomial-time functions efficiently invertible functions $\mathsf{enc}_{\mathsf{str}}$ and $\mathsf{enc}_{\mathsf{inst}}$ such that for $\mathsf{s} \leftarrow \mathsf{enc}_{\mathsf{str}}(F)$ and $\mathsf{u} \leftarrow \mathsf{enc}_{\mathsf{inst}}((x, y))$ we have that $(\mathsf{s}, \mathsf{u}, \mathsf{w}) = \mathsf{enc}(F, (x, y), w)$ for some w.
3. Monotonicity: Given $|F| \leq |G|$ we have that $|\mathsf{enc}_{\mathsf{str}}(F)| \leq |\mathsf{enc}_{\mathsf{str}}(G)|$.
4. Default instances: There exists $(\mathsf{u}_{\perp}, \mathsf{w}_{\perp})$ such that for any public parameters pp and structure s, we have that $(\mathsf{pp}, \mathsf{s}, \mathsf{u}_{\perp}, \mathsf{w}_{\perp}) \in \mathcal{R}_1$.

Assumption 5.1 (Non-Interactive Folding Scheme for Relaxed R1CS). There exists a non-interactive folding scheme for committed relaxed R1CS (Definition 4.15) in the plain model.

Justification. By applying the Fiat-Shamir transformation (Construction 7.1) to the interactive folding scheme for committed relaxed R1CS (Construction 4.1), we obtain a non-interactive multi-folding scheme for committed relaxed R1CS in the random oracle model. By instantiating the random oracle with a cryptographic hash function, we heuristically obtain a non-interactive multi-folding scheme for committed relaxed R1CS in the plain model. □

Lemma 5.1 (IVC-Compatibility of Folding R1CS). The non-interactive folding scheme for committed relaxed R1CS (Construction 4.1, Assumption 5.1) is IVC-compatible.

Proof. Gennero et al. [78] prove that R1CS is an NP-complete relation. By extension, this means that relaxed R1CS is an NP-complete relation. Moreover, we have that the (relaxed) R1CS structure matrices (A, B, C) can be invertibly derived given only F, and that the R1CS inputs and outputs vector, x, can be invertibly derived given the only the inputs and outputs of F.

Moreover, committed relaxed R1CS satisfies the default instances property. In particular, for arbitrary public parameters $(\mathsf{pp}_W, \mathsf{pp}_E)$ and structure (A, B, C) for instance $\mathsf{u}_{\perp} \leftarrow (0, 0, \overline{0}, \overline{0})$ and witness $\mathsf{w}_{\perp} \leftarrow (\mathbf{0}, 0, \mathbf{0}, 0)$ where $\overline{0}$ represents the identity in the commitment space and $\mathbf{0}$ represents the (appropriately sized) zero vector we have that

$$((\mathsf{pp}_W, \mathsf{pp}_E), (A, B, C), u_{\perp}, w_{\perp})$$

is a satisfying instance-witness pair in committed relaxed R1CS. □

Assumption 5.2 (Non-Interactive Folding Scheme for CCS). There exists a non-interactive folding scheme of type $\mathcal{R}^{\mu}_{\mathsf{LCCCS}} \times \mathcal{R}^{\nu}_{\mathsf{CCCS}} \to \mathcal{R}_{\mathsf{LCCCS}}$ (Definition 4.15) in the plain model.

Justification. We apply the Fiat-Shamir transformation (Construction 7.1) to the interactive folding scheme for CCS (Theorem 4.2) and instantiate the random oracle with a cryptographic hash function. □

Lemma 5.2 (IVC-Compatibility of Folding CCS). The non-interactive folding scheme for CCS (Theorem 4.2, Assumption 5.2) is IVC-compatible.

Proof (Sketch). This follows from the NP-completeness of CCS and because default linearized CCS instances can be appropriately generated by zeroing the instance and witness. Kothapalli and Setty [99] provide a formal proof. □

5.1.4 Construction

We now describe our formal construction for IVC using any IVC-compatible folding scheme.

Construction 5.1 (IVC). Consider a relation $\mathcal{R}_1$ and a committed relation $\mathcal{R}_2$ for a commitment scheme $(\mathsf{Com}, \mathsf{Gen})$. Let FS be an IVC-compatible non-interactive folding scheme of type $\mathcal{R}_1 \times \mathcal{R}_2 \to \mathcal{R}_1$. Let $(\mathsf{u}_\perp, \mathsf{w}_\perp)$ be a default instance-witness pair for $\mathcal{R}_1$ that satisfies any structure and public parameters. We construct an IVC scheme as follows.

Consider a polynomial-time function F that takes non-deterministic input and a succinct commitment scheme $(\mathsf{gen}, \mathsf{com})$. We begin by defining an augmented function F' as follows, where all input arguments are taken as non-deterministic advice.

$\underline{F'((\mathsf{pp}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{FS}}), \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi) \to \mathsf{x}\text{:}}$

1. If $i = 0$:
   (a) Check that $z_0 = z_i$.
   (b) Let $\mathsf{U}_{i+1} \leftarrow \mathsf{u}_\perp$.
2. Otherwise:
   (a) Parse u_i as (C, u'_i), a commitment to the witness and the remainder.
   (b) Check that u'_i references U_i in the output of the prior iteration of F'.

$$\mathsf{u}'_i \stackrel{?}{=} \mathsf{enc}_{\mathsf{inst}}(\mathsf{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}_{\mathsf{FS}}, i, z_0, z_i, \mathsf{U}_i)))$$

   (c) Compute a new instance which can be checked in place of u_i and U_i.

$$\mathsf{U}_{i+1} \leftarrow \mathsf{FS}.\mathcal{V}(\mathsf{vk}_{\mathsf{FS}}, \mathsf{U}_i, \mathsf{u}_i, \pi)$$

   (d) Output $\mathsf{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}_{\mathsf{FS}}, i + 1, z_0, F(z_i, \omega_i), \mathsf{U}_{i+1}))$

Given the augmented function F', we define the IVC proof relation Proof as follows.

$\underline{\mathsf{Proof}((\mathsf{pp_{FS}}, \mathsf{pp_{com}}), F, (i, z_0, z_i), \Pi_i) \to \{0,1\}}$:

1. If $i = 0$, check that $z_i = z_0$.
2. Otherwise:
   (a) Compute $\mathsf{s} \leftarrow \mathsf{enc_{str}}(F')$.
   (b) Compute $\mathsf{vk_{FS}} \leftarrow \mathsf{FS}.\mathcal{K}(\mathsf{pp_{FS}}, \mathsf{s})$.
   (c) Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$
   (d) Parse u_i as (C, u'_i). Check that $\mathsf{u}'_i = \mathsf{enc_{inst}}(\mathsf{com}(\mathsf{pp_{com}}, (\mathsf{vk_{FS}}, i, z_0, z_i, \mathsf{U}_i)))$.
   (e) Check that $(\mathsf{pp}, \mathsf{s}, \mathsf{U}_i, \mathsf{W}_i) \in \mathcal{R}_1$ and $(\mathsf{pp}, \mathsf{s}, \mathsf{u}_i, \mathsf{w}_i) \in \mathcal{R}_2$.

Next, we define the zero-interactive reduction $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

$\underline{\mathcal{G}(\lambda, n) \to \mathsf{pp}}$:

1. Output $(\mathsf{pp_{FS}}, \mathsf{pp_{com}}) \leftarrow (\mathsf{FS}.\mathcal{G}(\lambda, n), \mathsf{gen}(\lambda, n))$.

$\underline{\mathcal{K}(\mathsf{pp}, F) \to (\mathsf{pk}, \mathsf{vk})}$:

1. Compute $\mathsf{s} \leftarrow \mathsf{enc_{str}}(F')$, $(\mathsf{pk_{FS}}, \mathsf{vk_{FS}}) \leftarrow \mathsf{FS}.\mathcal{K}(\mathsf{pp_{FS}}, \mathsf{s})$, $\mathsf{pk} \leftarrow (F, \mathsf{pk_{FS}}, (\mathsf{pp}, \mathsf{vk_{FS}}, \mathsf{s}))$.
2. Output the prover and verifier key $(\mathsf{pk}, \perp)$.

$\underline{\mathcal{P}(\mathsf{pk}, ((z_i, z_{i+1}), \omega_i) \in \mathsf{Eval}, ((i, z_0, z_i), \Pi_i) \in \mathsf{Proof}) \to ((i+1, z_0, z_{i+1}), \Pi_{i+1}) \in \mathsf{Proof}}$:

1. Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$
2. If $i = 0$, let
$$(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow (\mathsf{u}_\perp, \mathsf{w}_\perp, \perp)$$
Otherwise, compute a new instance-witness pair which can be checked in place of $(\mathsf{U}_i, \mathsf{W}_i)$ and $(\mathsf{u}_i, \mathsf{w}_i)$:
$$(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow \mathsf{FS}.\mathcal{P}(\mathsf{pk}, (\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i)).$$
3. Compute $y \leftarrow F'((\mathsf{pp_{com}}, \mathsf{vk_{FS}}), \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi)$
4. Compute an instance-witness pair encoding the valid execution of F'
$$(\mathsf{u}'_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{enc}(F', (((\mathsf{pp_{com}}, \mathsf{vk_{FS}}), \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi), y), \omega_i).$$
5. Compute the committed instance
$$\mathsf{u}_{i+1} \leftarrow (\mathsf{u}'_{i+1}, \mathsf{Commit}(\mathsf{pp_{FS}}, \mathsf{w}_{i+1})).$$

6. Let $\Pi_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}))$ and output $((i+1, z_0, z_{i+1}), \Pi_{i+1})$.

$\underline{\mathcal{V}(\mathsf{vk}, (z_i, z_{i+1}), (i, z_0, z_i))}$:

1. Output $(i+1, z_0, z_{i+1})$.

Theorem 5.1 (IVC). Construction 5.1 is an IVC scheme.

Proof. This follows from Lemma 5.3, and Lemma 5.4. □

Lemma 5.3 (Completeness). Construction 5.1 is complete.

Proof. Consider arbitrary PPT adversary $\mathcal{A}$. Let $\mathsf{pp} = (\mathsf{pp}_{\mathsf{FS}}, \mathsf{pp}_{\mathsf{com}}) \leftarrow \mathcal{G}(\lambda, n)$. Suppose on input pp the adversary $\mathcal{A}$ picks structure F, picks a satisfying Proof instance (i, z_0, z_i) and corresponding witness $\Pi_i = ((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$, and a satisfying Eval instance (z_i, z_{i+1}) and corresponding witness ω_i such that

$$(\mathsf{pp}, F, ((z_i, z_{i+1}), (i, z_0, z_i)), (\omega_i, \Pi_i)) \in \mathsf{Eval} \overset{\sim_1}{\times} \mathsf{Proof}.$$

We must show that for $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F)$

$$\langle \mathcal{P}, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), ((z_i, z_{i+1}), (i, z_0, z_i)), (\omega_i, \Pi_i)) \in \mathsf{Proof}.$$

Indeed, in the case where $i = 0$, the prover computes

$$(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow (\mathsf{u}_\perp, \mathsf{w}_\perp, \perp)$$

Alternatively, if $i \geq 1$, the prover computes

$$(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow \mathsf{FS}.\mathcal{P}(\mathsf{pp}_{\mathsf{FS}}, (s, u_i, w_i), (s, U_i, W_i))$$

Next, the prover computes

$$y \leftarrow F'((\mathsf{pp}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{FS}}), \mathsf{U}_i, \mathsf{u}_i, (i, z_0, z_i), \omega_i, \pi).$$

When $i = 0$, the prover can compute a valid output y because we must have that $z_0 = z_i$ by the precondition. Alternatively, if $i \geq 1$, the prover can still compute a valid output y because we must have that

$$\mathsf{enc}_{\mathsf{io}}(\mathrm{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}, i, z_0, z_i, U_i))) = \mathsf{u}'_i$$

by the precondition.

By construction, F' outputs

$$y \leftarrow \mathrm{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}_{\mathsf{FS}}, i+1, z_0, F(z_i, \omega_i), \mathsf{U}_{i+1})).$$

where

$$\mathsf{U}_{i+1} \leftarrow \perp$$

if $i = 0$ and

$$\mathsf{U}_{i+1} \leftarrow \mathsf{FS}.\mathcal{V}(\mathsf{vk}_{\mathsf{FS}}, \mathsf{U}_i, \mathsf{u}_i, \pi)$$

otherwise. Next, $\mathcal{P}$ computes the corresponding encoded instance-witness pair $(\mathsf{u}'_{i+1}, \mathsf{w}_{i+1})$ in $\mathcal{R}'_2$ and appends the commitment to w_{i+1} to the partial instance u'_{i+1} to get a satisfying instance-witness pair $(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$ in $\mathcal{R}_2$. Next, the prover outputs the instance

$$(i + 1, z_0, z_{i+1})$$

and the witness

$$\Pi_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), \mathsf{u}_{i+1}, \mathsf{w}_{i+1}).$$

We must argue that the prover's output is a satisfying instance-witness pair in Proof. Indeed, by construction, the output of F' is $\mathsf{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}_{\mathsf{FS}}, i + 1, z_0, z_{i+1}, \mathsf{U}_{i+1}))$. Therefore, by construction of $\mathcal{P}$, we have that

$$\mathsf{u}'_{i+1} = \mathsf{enc}(F', \mathsf{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}, i + 1, z_0, z_{i+1}, \mathsf{U}_{i+1})))$$

Next, by the completeness of enc, we have that

$$(\mathsf{s}, \mathsf{u}'_{i+1}, \mathsf{w}_{i+1}) \in \mathcal{R}'_2.$$

Therefore, by construction of $\mathcal{P}$, we have that

$$(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, \mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \in \mathcal{R}_2.$$

Next, in the case where $i = 0$, because we have that $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}) \leftarrow (\mathsf{u}_\perp, \mathsf{w}_\perp)$, by the required property of default instances we have that

$$(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, \mathsf{U}_{i+1}, \mathsf{W}_{i+1}) \in \mathcal{R}_1.$$

Alternatively, in the case where $i \geq 1$, by the precondition that $(\mathsf{U}_i, \mathsf{W}_i)$ and $(\mathsf{u}_i, \mathsf{w}_i)$ are satisfying, and by the correctness of FS, we have that

$$(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, \mathsf{U}_{i+1}, \mathsf{W}_{i+1}) \in \mathcal{R}_1.$$

Therefore, we have that the prover's output instance-witness pair is satisfying. □

Lemma 5.4 (Knowledge Soundness). Construction 5.1 is knowledge sound.

Proof. Consider arbitrary expected-polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$. Suppose $\mathsf{pp} = (\mathsf{pp}_{\mathsf{FS}}, \mathsf{pp}_{\mathsf{com}}) \leftarrow \mathcal{G}(\lambda, n)$. Suppose on input pp the adversary $\mathcal{A}$ picks structure F, picks a Proof instance (i, z_0, z_i), an Eval instance (z_i, z_{i+1}) and corresponding auxiliary state st. Suppose now that for $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F)$

$$\Pr[(\mathsf{pp}, \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), ((F, i, z_0, z_i), (F, z_i, z_{i+1})), \mathsf{st})) \in \mathsf{Proof}] = \epsilon.$$

We must construct an expected-polynomial-time extractor $\mathcal{E}$ such that

$$\Pr[(\mathsf{pp}, u_1, \mathcal{E}(\mathsf{pp}, F, ((z_i, z_{i+1}), (i, z_0, z_i)), \mathsf{st})) \in \mathsf{Eval} \overset{\sim 1}{\times} \mathsf{Proof}] = \epsilon - \mathsf{negl}(\lambda)$$

where $\sim_1$ is defined as in Definition 5.3.

At a high level, using $\mathcal{P}^*$ we design an adversary $\mathcal{P}^*_{\mathsf{FS}}$ for the underlying folding scheme that succeeds with probability ϵ. By the knowledge-soundness of the underlying folding scheme we have that there exist a corresponding extractor $\mathcal{E}_{\mathsf{FS}}$ that succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$. We then use the extractor $\mathcal{E}_{\mathsf{FS}}$ to construct the desired extractor $\mathcal{E}$ for the IVC scheme.

We start by constructing the adversaries $\mathcal{A}_{\mathsf{FS}}$ and $\mathcal{P}^*_{\mathsf{FS}}$ for the underlying folding scheme.

$\underline{\mathcal{A}_{\mathsf{FS}}(\mathsf{pp}_{\mathsf{FS}}) \to (\mathsf{s}, (\mathsf{U}_i, \mathsf{u}_i), \mathsf{st})}$

1. Compute $\mathsf{pp}_{\mathsf{com}} \leftarrow \mathsf{gen}(\lambda, n)$ and let $\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{FS}}, \mathsf{pp}_{\mathsf{com}})$.
2. Compute $(F, ((i, z_0, z_i), (z_i, z_{i+1})), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$.
3. Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, F)$.
4. Compute $\mathsf{s} \leftarrow \mathsf{enc}_{\mathsf{str}}(F')$.
5. Compute $((i+1, z_0, z_{i+1}), \Pi_{i+1}) \leftarrow \mathcal{P}^*(\mathsf{pk}, ((i, z_0, z_i), (z_i, z_{i+1})), \mathsf{st})$.
6. Parse Π_{i+1} as $((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}))$
7. Parse w_{i+1} for the inputs to the folding verifier in F':
$$(\mathsf{U}_i, \mathsf{u}_i, \pi) \leftarrow \mathsf{enc}^{-1}(\mathsf{s}, \mathsf{u}_{i+1}, \mathsf{w}_{i+1}).$$
8. Let $\mathsf{st} \leftarrow (\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi)$.
9. Output $(\mathsf{s}, (\mathsf{U}_i, \mathsf{u}_i), \mathsf{st})$.

$\underline{\mathcal{P}^*_{\mathsf{FS}}(\mathsf{pk}, (\mathsf{U}_i, \mathsf{u}_i), \mathsf{st}) \to (\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi)}$

1. Parse st as $(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi)$ and s from pk.
2. Output $(\mathsf{s}, \mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi)$.

We now analyze the success probability of $\mathcal{P}^*_{\mathsf{FS}}$. By the precondition, we have that

$$(\mathsf{pp}, F, (i+1, z_0, z_{i+1}), \Pi_{i+1}) \in \mathsf{Proof}$$

with probability ϵ. Then, because

$$((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}))$$

was parsed from Π_{i+1}, by definition of relation Proof, we have that

$$(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, \mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \in \mathcal{R}_2$$

and

$$(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, \mathsf{U}_{i+1}, \mathsf{W}_{i+1}) \in \mathcal{R}_1 \tag{5.1}$$

with probability ϵ. Then, because we have that $(\mathsf{U}_i, \mathsf{u}_i, \pi_{\mathsf{FS}})$ are inputs to the folding verifier parsed from w_{i+1} and that the output of F' is

$$\mathsf{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}_{\mathsf{FS}}, i+1, z_0, z_{i+1}, \mathsf{U}_{i+1}))$$

we have that on input $(\mathsf{U}_i, \mathsf{u}_i, \pi)$ the folding verifier outputs U_{i+1} by the binding property of com. Then, for

$$(\mathsf{s}, (\mathsf{U}_i, \mathsf{u}_i), \mathsf{st}) \leftarrow \mathcal{A}_{\mathsf{FS}}(\mathsf{pp}_{\mathsf{FS}})$$

by Equation (5.1) we have that $\mathcal{P}^*_{\mathsf{FS}}$ succeeds with probability ϵ. Formally, we have that for $(\mathsf{pk}_{\mathsf{FS}}, \mathsf{vk}_{\mathsf{FS}}) \leftarrow \mathcal{K}(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s})$

$$\Pr[(\mathsf{pp}_{\mathsf{FS}}, \langle \mathcal{P}^*_{\mathsf{FS}}, \mathsf{FS}.\mathcal{V}\rangle((\mathsf{pk}_{\mathsf{FS}}, \mathsf{vk}_{\mathsf{FS}}), (\mathsf{s}, (\mathsf{U}_i, \mathsf{u}_i)), \mathsf{st})) \in \mathcal{R}_1] = \epsilon.$$

Then, by the knowledge-soundness of the underlying folding scheme, we have that there exists a corresponding extractor $\mathcal{E}_{\mathsf{FS}}$ that succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$. Formally, we have that

$$\Pr[(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, (\mathsf{U}_i, \mathsf{u}_i), \mathcal{E}_{\mathsf{FS}}(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, (\mathsf{U}_i, \mathsf{u}_i), \mathsf{st})) \in \mathcal{R}_1 \times \mathcal{R}_2] = \epsilon - \mathsf{negl}(\lambda). \tag{5.2}$$

Using $\mathcal{E}_{\mathsf{FS}}$, we now construct the desired extractor $\mathcal{E}$.

$\underline{\mathcal{E}(\mathsf{pp}, F, ((z_i, z_{i+1}), (i, z_0, z_i)), \mathsf{st}) \to (\Pi_i, \omega_i)}$

1. Parse the public parameters pp as $(\mathsf{pp}_{\mathsf{FS}}, \mathsf{pp}_{\mathsf{com}})$.
2. Compute
$$((i+1, z_0, z_{i+1}), \Pi_{i+1}) \leftarrow \mathcal{P}^*(\mathsf{pp}, ((F, i, z_0, z_i), (F, z_i, z_{i+1})), \mathsf{st})$$
3. Parse Π_{i+1} as
$$((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}))$$
4. Parse w_{i+1} for the inputs to the folding verifier in F' and the non-deterministic input for F
$$(\mathsf{U}_i, \mathsf{u}_i, \pi, \omega_i) \leftarrow \mathsf{enc}^{-1}(\mathsf{s}, \mathsf{u}_{i+1}, \mathsf{w}_{i+1})$$
5. Compute
$$\mathsf{st}' \leftarrow (\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi)$$
6. Compute
$$(\mathsf{W}_i, \mathsf{w}_i) \leftarrow \mathcal{E}_{\mathsf{FS}}(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, (\mathsf{U}_i, \mathsf{u}_i), \mathsf{st}')$$
7. Compute $\Pi_i \leftarrow ((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$ and output (Π_i, ω_i).

We now analyze the success probablity of $\mathcal{E}$. In the base case, when $i = 0$ we observe that because w_{i+1} is accepting with probability ϵ, we must have that the checks of F' pass with the same probablity, and therefore the only requirement that $z_0 = z_i$ holds with the same probability. By observation we have that $\mathcal{E}$ provides the same distribution of inputs to $\mathcal{E}_{\mathsf{FS}}$ as $\mathcal{A}_{\mathsf{FS}}$. Therefore, by Equation (5.2), we have that

$$(\mathsf{pp}_{\mathsf{FS}}, (\mathsf{s}, (\mathsf{U}_i, \mathsf{u}_i)), (\mathsf{W}_i, \mathsf{w}_i)) \in \mathcal{R}_1 \times \mathcal{R}_2$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. Therefore, by definition we have that

$$(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, \mathsf{U}_i, \mathsf{W}_i) \in \mathcal{R}_1$$
$$(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, \mathsf{u}_i, \mathsf{w}_i) \in \mathcal{R}_2$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. Moreover, by the success probability of $\mathcal{P}^*$, we have that

$$(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}, \mathsf{u}_{i+1}, \mathsf{w}_{i+1}) \in \mathcal{R}_2 \tag{5.3}$$

with probability ϵ. Then, because U_i and u_i are parsed from w_{i+1}, by construction of F' and by the precondition, we must have that

$$\mathsf{enc}(F', \mathrm{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}_{\mathsf{FS}}, i, z_0, z_i, \mathsf{U}_i))) = \mathsf{u}'_i$$

where u'_i denotes the portion of u_i that excludes to the commitment to the witness. Therefore, for $\Pi_i \leftarrow ((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i))$ we have that

$$(\mathsf{pp}, F, (i, z_0, z_i), \Pi_i) \in \mathsf{Proof} \tag{5.4}$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. Moreover, because ω_i was parsed from w_{i+1} By Equation (5.3), we have that

$$((F, (z_i, z_{i+1})), \omega_i) \in \mathsf{Eval} \tag{5.5}$$

with probability $\epsilon - \mathsf{negl}(\lambda)$. Thus, by Equations (5.4) and (5.5) we have that

$$\Pr[(\mathsf{pp}, ((z_i, z_{i+1}), (i, z_0, z_i)), \mathcal{E}(\mathsf{pp}, F, ((z_i, z_{i+1}), (i, z_0, z_i)), \mathsf{st})) \in \mathsf{Eval} \overset{\sim}{\times}^1 \mathsf{Proof}] = \epsilon - \mathsf{negl}(\lambda).$$

□

5.1.5 Implementation and Evaluation

For a concrete performance analysis, we instantiate our IVC scheme (Construction 5.1) with the folding scheme for R1CS (Construction 4.1) instantiated with the Pedersen commitment scheme [119]. We refer to the resulting IVC proof system as Nova.

The following theorem captures the cryptographic and efficiency characteristics of our folding scheme for relaxed R1CS.

Theorem 5.2 (Folding R1CS, Efficiency). Construction 4.1 is a constant-round, public-coin, zero-knowledge folding scheme for relaxed R1CS where for N-sized relaxed R1CS instances over a finite field $\mathbb{F}$ with the same "structure" (i.e., R1CS coefficient matrices), the prover's work is $O_\lambda(N)$, and the verifier's work and the communication are both $O_\lambda(1)$, assuming the existence of any additively-homomorphic commitment scheme that provides $O_\lambda(1)$-sized commitments to N-sized vectors over $\mathbb{F}$ (e.g., Pedersen's commitments), where λ is the security parameter.

Our folding scheme for R1CS remains one of the most efficient folding schemes for NP. As a result, Nova achieves the smallest recursion overhead in the literature. Since the verifier's costs in the non-interactive version of the folding scheme for relaxed R1CS is $O_\lambda(1)$, the size of the computation that Nova's prover proves at each incremental step is $\approx|F|$, assuming N-sized vectors are committed with an $O_\lambda(1)$-sized commitments (e.g., Pedersen's commitments). In particular, the verifier circuit in Nova is constant-sized and its size is dominated by two *group scalar multiplications*. Furthermore, Nova's prover's work at each step is dominated by two multiexponentiations of size $\approx|F|$. Note that Nova's prover does not perform any FFTs, so it can be instantiated efficiently using *any* cycles of elliptic curves where the discrete-logarithm problem is hard.

With the description thus far, the size of an IVC proof is $O_\lambda(|F|)$. Instead of sending such a proof to a verifier, at any point in the incremental computation, Nova's prover can prove the knowledge of a satisfying witness to the running relaxed R1CS instance in zero-knowledge with an $O_\lambda(\log|F|)$-sized succinct proof using a zkSNARK that we design by adapting Spartan [124]. Kothapalli, Setty, and Tzialla [101] provide details of this construction. The following theorem summarizes the resulting efficiency characteristics.

Theorem 5.3 (Nova, Efficiency). For any incremental function where each step of the incremental function applies a (non-deterministic) function F, there exists an IVC scheme with the following efficiency characteristics, assuming N-sized vectors are committed with an $O_\lambda(1)$-sized commitments.

- IVC proof sizes are $O(|F|)$ and the verifier's work to verify them is $O_\lambda(|F|)$. The prover's work at each incremental step is $\approx|F|$. Specifically, the prover's work at each step is dominated by two multiexponentiations of size $\approx|F|$.
- Succinct zero-knowledge proofs of valid IVC proofs are size $O_\lambda(\log|F|)$, and the verifier's work to verify them is either $O_\lambda(\log|F|)$ or $O_\lambda(|F|)$ depending on the commitment scheme for vectors. The prover's work to produce this succinct zero-knowledge proof is $O_\lambda(|F|)$.

Comparison with Prior Work

Figure 5.1 compares Nova with prior approaches. Nova's approach can be viewed as taking Halo, due to Bowe, Grigg, and Hopwood [38], to its logical conclusion. Specifically:

- At each incremental step, Halo's verifier circuit verifies a "partial" SNARK. This still requires Halo's prover to perform $|F|$-sized FFTs and $O(|F|)$ exponentiations (i.e.,

not an $|F|$-sized multiexponentiation). Whereas, in Nova, the verifier circuit folds an entire NP instance representing computation at the prior step into a running relaxed R1CS instance. This only requires Nova's prover to commit to a satisfying assignment of an $\approx|F|$-sized circuit (which computes F and performs the verifier's computation in a folding scheme for relaxed R1CS), so at each step, Nova's prover only computes an $O(|F|)$-sized multiexponentiation and does not compute any FFTs. So, Nova's prover incurs lower costs than Halo's prover, both asymptotically and concretely.

- The verifier circuit in Halo is of size $O_\lambda(\log |F|)$ whereas in Nova, it is $O_\lambda(1)$. Concretely, the dominant operations in Halo's circuit is $O(\log |F|)$ group scalar multiplications, whereas in Nova, it is two group scalar multiplications.

- Halo and Nova have the same proof sizes $O_\lambda(\log |F|)$ and verifier time $O_\lambda(|F|)$.

Subsequently, Bünz et al. [43] apply Halo's approach to other polynomial commitment schemes. Halo Infinite [35] generalizes the approach in Halo [38] to any homomorphic polynomial commitment scheme; they also obtain PCD (and hence IVC) even when polynomial commitment schemes do not satisfy succinctness.

Furthermore, Bünz et al. [45] propose a variant of the approach in Halo, where they realize proof-carrying data [30] (a generalization of IVC for any recursive topology) without relying on succinct proofs. Specifically, they first devise a non-interactive argument of knowledge (NARK) for R1CS with $O_\lambda(N)$-sized proofs and $O_\lambda(N)$ verification times for N-sized R1CS instances. Then, they show that most of the NARK's verifier's computation can be deferred by performing $O_\lambda(1)$ work in the verifier circuit. For zero-knowledge, Nova relies on zero-knowledge proofs with succinct proofs, whereas their approach does not rely on succinct proofs. However, Nova's approach has several efficiency advantages over the work of Bünz et al [45]:

- Their prover's work for multiexponentiations at each step is roughly $4\times$ higher and the size of their verifier circuit is roughly $2\times$ larger than Nova.

- Proof sizes are $O_\lambda(|F|)$ in their work, whereas in Nova, they are $O_\lambda(\log |F|)$. In theory, they can also compress their proofs, using a succinct proof, but unlike Nova, they do not specify how to do so in a concretely efficient manner.

Implementation and Performance Evaluation

We implement Nova as a library in about 6,000 lines of Rust [3], which is currently being actively maintained. The library is generic over a cycle of elliptic curves and a hash function (used internally as the random oracle). The library provides candidate implementations with the Pasta cycle of elliptic curves [4] and Poseidon [2, 85]. For the former, Nova relies on `pasta-msm` [5], a high-performance library for computing multiexponentiations over the Pasta cycle of curves. Finally, the library accepts F (i.e., a step of the incremental computation) as a bellperson gadget [1].

	"Verifier circuit" (dominant ops)	Prover (each step)	Proof size	Verifier	assumptions
BCTV14 [27] with [86]†	3 $\mathbb{P}$	$O(C)$ FFT $O(C)$ MSM	$O_\lambda(1)$	$O_\lambda(1)$	q-type
Spartan [124]-based IVC	$O(\sqrt{C})$ $\mathbb{G}$	$O(C)$ MSM	$O_\lambda(\sqrt{C})$	$O_\lambda(\sqrt{C})$	DLOG, RO
Fractal [55]	$O_\lambda(\log^2 C)$ $\mathbb{F}$ $O(\log^2 C)$ $\mathbb{H}$	$O(C)$ FFT $O(C)$ MHT	$O_\lambda(\log^2 C)$	$O_\lambda(\log^2 C)$	RO
Halo [38]	$O(\log C)$ $\mathbb{G}$	$O(C)$ FFT $O(C)$ EXP	$O_\lambda(\log C)$	$O_\lambda(C)$	DLOG, RO
BCLMS [43]⋆	8 $\mathbb{G}$	$O(C)$ FFT $O(C)$ MSM	$O_\lambda(C)$	$O_\lambda(C)$	DLOG, RO
Nova (this work)	2 $\mathbb{G}$	$O(C)$ MSM	$O_\lambda(\log C)$	$O_\lambda(C)$	DLOG, RO
Nova (this work)	2 $\mathbb{G}_T$	$O(C)$ MSM	$O_\lambda(\log C)$	$O_\lambda(\log C)$	SXDH, RO

† Requires per-circuit trusted setup and is undesirable in practice
$O(C)$ FFT: FFT over an $O(C)$-sized vector costing $O(C \log C)$ operations over $\mathbb{F}$
$O(C)$ MHT: Merkle tree over an $O(C)$-sized vector costing $O(C)$ hash computations
$O(C)$ EXP: $O(C)$ exponentiations in a cryptographic group
$O(C)$ MSM: $O(C)$-sized multi-exponentiation in a cryptographic group

Figure 5.1: Asymptotic costs of Nova and its baselines to produce and verify a proof for an incremental computation where each incremental step applies a function F. C denotes the size of the computation at each incremental step, i.e., $|F|+|\mathcal{C}_\mathcal{V}|$, where $\mathcal{C}_\mathcal{V}$ is the "verifier circuit" in IVC. The "verifier circuit" column depicts the number of dominant operations in $\mathcal{C}_\mathcal{V}$, where $\mathbb{P}$ denotes a pairing in a pairing-friendly group, $\mathbb{F}$ denotes the number of finite field operations, $\mathbb{H}$ denotes a hash computation, and $\mathbb{G}$ denotes a scalar multiplication in a cryptographic group. The prover column depicts the cost to the prover for each step of the incremental computation, and proof sizes and verifier times refer respectively to the size of the proof of the incremental computation and the associated verification times. For Nova's proof sizes and verification times, we depict the compressed proof sizes (otherwise, they are $O_\lambda(C)$) and the time to verify a compressed proof (otherwise, they are $O_\lambda(C)$). Rows with RO require heuristically instantiating the random oracle with a concrete hash function in the standard model.

	Primary Curve	Secondary Curve
Scalar multiplications	12,362	12,362
Random oracle call	1,431	1,434
Collision-resistant hash	2,300	2,306
Non-native arithmetic	3,240	3,240
Glue code	1,251	1,782
Total	20,584	21,124

Figure 5.2: A detailed breakdown of sub-routines in Nova's verifier's circuit and the associated number of R1CS constraints. The verifier circuit on each of the curves in the cycle are not identical as they have slightly different base cases. We find that a majority of constraints in the verifier circuit step from the group scalar multiplications.

Recursion Overheads. We measure the size of Nova's verifier circuit, as it determines the *recursion overhead*: the number of additional constraints that the prover must prove at each incremental step besides proving an invocation of F. We find that Nova's verifier

circuit is roughly 20000 R1CS constraints (Figure 5.2). This is the smallest verifier circuit in the literature and hence Nova incurs the lowest recursion overhead. Specifically, Nova's recursion overhead is $> 10\times$ lower than in SNARK-based IVC [27] with state-of-the-art per-circuit trusted setup SNARK [86], and over $100\times$ smaller than with a SNARK without trusted setup [55]. Compared to recent works, Nova's recursion overhead is over $7\times$ lower than Halo's [38], and over $2\times$ lower than the scheme of Bünz et al. [43].

Performance of Nova. We experiment with Nova on an Azure Standard F32s_v2 VM (16 physical CPUs, 2.70 GHz Intel(R) Xeon(R) Platinum 8168, and 64 GB memory). In our experiments, we vary the number of constraints in F. Our performance metrics are: the prover time, the verifier time, and proof sizes. We measure these for Nova's IVC scheme as well as its Spartan-based zkSNARK [101] to compress IVC proofs. Figure 5.3 depicts our results, and we find the following.

- The prover's per-step cost to produce an IVC proof and compress it scale sub-linearly with the size of F (since the cost is dominated by two multiexponentiations, which scale sub-linearly due to the Pippenger algorithm and parallelize better at larger sizes). When $|F| \approx 2^{20}$ constraints, the prover's per-step cost to produce an IVC proof is $\approx 1\mu$s/constraint. For the same F, the cost to produce a compressed IVC proof is $\approx 24\mu$s/constraint. If the prover produces a compressed IVC proof every ≈ 24 steps, the prover incurs at most $2\times$ overhead to compress IVC proofs. Similarly, if the prover compresses its IVC proof every ≈ 240 steps, the overhead drops to $\approx 20\%$.
- Compressed IVC proofs are $\approx$ 8–9 KB and are significantly shorter than IVC proofs (e.g., they are $\approx 7{,}400\times$ shorter when $|F| \approx 2^{20}$ constraints).
- Verifying a compressed proof is only $\approx 2\times$ higher costs than verifying a significantly longer IVC proof.

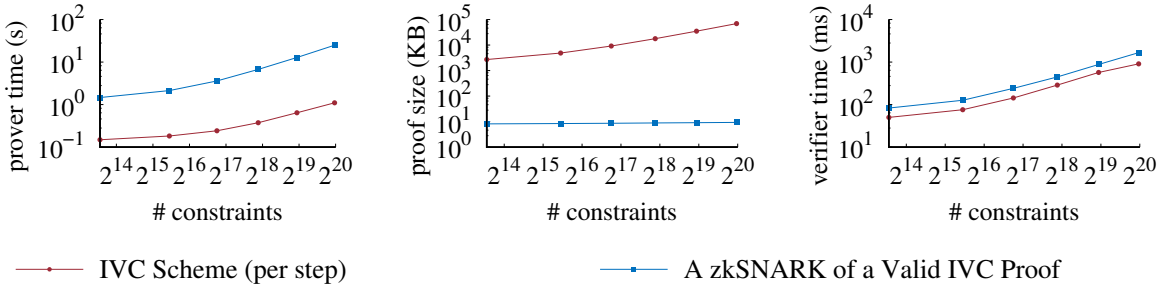


Figure 5.3: Performance of Nova as a function of $|F|$. See the text for details.

5.2 Non-Uniform Incrementally Verifiable Computation

Recall that a central application of IVC is the ability to prove virtual machine execution (e.g., program executions on the Ethereum or WASM virtual machine). In particular,

using IVC we can recursively prove that "the current state of the virtual machine is valid" by proving that "there exists a proof for the previous state of the virtual machine and the most recent cycle is valid". The standard realization employs a *universal circuit* (e.g., [7, 23, 25, 80, 107]) that can execute any instruction supported by the machine as the IVC step function F. To prove the correct execution of programs on the corresponding machine, it suffices to prove repeated invocations of this circuit on an input program and memory state [27]. Unfortunately, the cost of proving a program's step is proportional to the size of the universal circuit (i.e., sum of sizes of circuits of all instructions supported by the machine)—even though the step invokes only one of the instructions.

Given the high costs imposed by universal circuits, designers of these machines aim to employ a minimal instruction set, to keep the size of the universal circuit and thereby the cost of proving a program step minimal [24, 25, 80]. However, this is a not a panacea: for real applications, we need to execute an enormous number of iterations of the minimal circuit (e.g., billions of iterations), making the prover's work largely untenable. This also means that emulating real programs that target *existing* virtual machines with rich instruction sets (e.g., EVM, RISC-V, or WASM) via a machine with a minimal instruction set would incur enormous costs.

We are thus interested in designing a system where the cost of proving a step of a program execution is proportional only to the size of the circuit representing the instruction invoked by the program step and independent of the circuit sizes of the uninvoked instructions. To model such a system, we introduce a generalization of IVC [133], called *non-uniform IVC (NIVC)*, to formally capture the desired cost profile.

In particular, consider a collection of $\ell + 1$ non-deterministic, polynomial-time computable functions $((F_1, \ldots, F_\ell), \varphi)$, where $\ell \geq 1$. Here, the functions F_i for $i \in [\ell]$ represent individual instructions and take a state and a non-deterministic input and produce a new state. The function φ represents a program, taking as input the same state and the non-deterministic input deciding which of these functions should be executed in a particular step. An NIVC scheme enables a prover to incrementally prove that it has performed an n-step computation with an initial input z_0 to produce an output z_n. In particular, at step i, the prover proves that it has applied F_j on input (z_{i-1}, ω_{i-1}) to produce an output z_i, where z_{i-1} is the output of step $i-1$, ω_{i-1} is a (potentially secret) non-deterministic input from the prover for step i, and $j = \varphi(z_{i-1}, \omega_{i-1})$. That is, φ selects one of the possible ℓ functions to apply at step i using inputs to step i. A bit more concisely, for a specified $((F_1, \ldots, F_\ell), \varphi)$ and (n, z_0, z_n), the prover proves the knowledge of a set of non-deterministic values $(\omega_0, \ldots, \omega_{n-1})$ and $(z_1, \ldots, z_{n-1})$ such that for all $i \in \{0, \ldots, n-1\}$, we have that $z_{i+1} = F_{\varphi(z_i, \omega_i)}(z_i, \omega_i)$. Crucially, the prover's work at step i is proportional only to $|F_j|$, where $j = \varphi(z_i, \omega_i)$, rather than $|F_1| + \ldots + |F_\ell|$.

Given a definition of NIVC, we provide a construction for NIVC using any IVC-compatible folding scheme (Definition 5.4). As with IVC, instead of directly proving the knowledge of a satisfying witness to some prescribed F_j for $j \in \{1, \ldots, \ell\}$ in each step, the prover proves the knowledge of a satisfying witness to an augmented function F'_j. The augmented function F'_j, in addition to running F_j, performs additional bookkeeping using a folding scheme to help verifiably update the NIVC proof.

At first glance, a straw-man approach is to have each F'_j take as input an instance

that claims the correct execution of the latest iteration and then fold that instance into a running instance using a folding scheme. However, known folding schemes require that both instances refer to the same computations in their structure matrices. In the case of standard IVC, as there is only one function that can be applied at each iterative step, this holds naturally. However, this is not the case for non-uniform IVC.

To address this, F'_j instead takes a list U_i of running instances, where $\mathsf{U}_i[j]$ attests to all prior iterations of F'_j up to $i-1$ steps. As such, checking all of U_i is equivalent to checking $i-1$ steps. In addition, F'_j takes as input a new instance u_i, which claims the correctness of the i'th step. Instead of directly checking this instance (which would be concretely expensive), F'_j folds u_i into the appropriate instance in U_i according to φ to produce a new list of running instances U_{i+1}. To claim the correctness of F'_j itself, the prover produces a new instance u_{i+1}.

We let the NIVC proof Π_i contain the list U_i, the fresh instance u_i, the corresponding witnesses and the instruction pc_i corresponding to the fresh instance. Thus, the prover can use parts of Π_i as input to the appropriate function F'_j to produce U_{i+1}, u_{i+1}, and pc_{i+1} and separately compute the corresponding witnesses. These terms together define Π_{i+1}. We overview this construction in Figure 5.4.

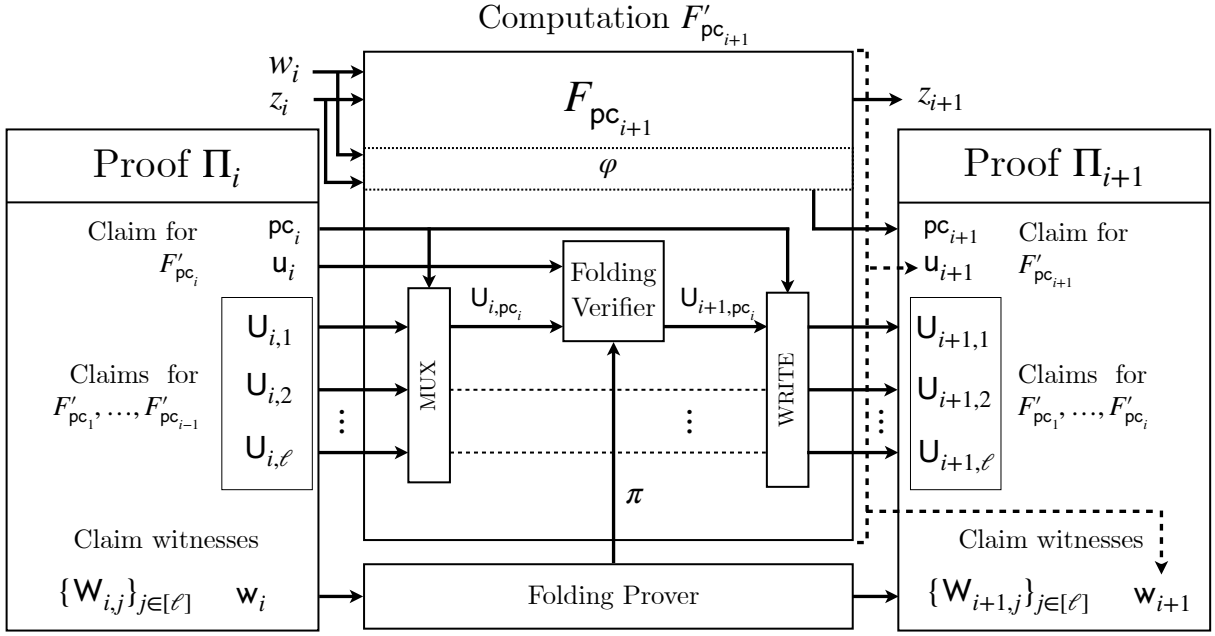


Figure 5.4: Overview of non-uniform IVC from folding.

5.2.1 Defining Non-Uniform IVC

This section introduces *non-uniform IVC (NIVC)*, a generalization of IVC, where at each step of an incremental computation, the prover proves the satisfiability of a relation chosen from a set of possible relations (the choice of which relation to use is made by an additional designated relation), whereas in the standard IVC, there is only one possible relation. As

a result of this generalization, the overall relation proven by non-uniform IVC can be a non-uniform circuit (i.e., circuits without repeating structure), which motivates its name. As detailed in the introduction, non-uniform IVC implies proofs of program executions on machines with a pre-defined custom instruction set. In the upcoming sections, we construct an efficient NIVC scheme.

Recall that in IVC, for a polynomial-time function F, the prover takes as input a claim (i, z_0, z) and a corresponding proof Π_i that attests to the knowledge of witnesses $(\omega_0, \ldots, \omega_{i-1})$ such that by computing $z_{j+1} \leftarrow F(z_j, \omega_j)$ for all $j \in \{0, \ldots, i-1\}$ we have that $z = z_i$. Given a new witness ω_i, the prover computes a new proof Π_{i+1} of the same size, which proves the statement $(i+1, z_0, z_{i+1})$ for $z_{i+1} = F(z_i, \omega_i)$.

In NIVC, we extend IVC to handle a number of arbitrary polynomial-time functions $(F_1, \ldots, F_\ell)$. The choice of which function F_j for $j \in [\ell]$ is executed at a particular step in the incremental computation is handled by an additional polynomial-time function φ. More specifically, NIVC captures an incremental proof system for the following augmented statement: There exists $(\omega_0, \ldots, \omega_{i-1})$ such that on initial input z_0 and claimed output z, by computing $z_{j+1} \leftarrow F_{\varphi(z_j, \omega_j)}(z_j, \omega_j)$ for all $j \in \{0, \ldots, i-1\}$, we have that $z = z_i$.

We adapt the above succinctness, completeness and knowledge soundness definitions of IVC for the setting of NIVC. Moreover, for NIVC to be a meaningful notion, we stipulate an additional efficiency requirement: the prover's work at each step scales only with the size of the function executed at that step. Without such a requirement, IVC immediately implies NIVC with the use of a single universal circuit that embeds all functions $(F_1, \ldots, F_\ell)$. Observe that if we fix $\ell = 1$ and that φ outputs 1, we recovers the definition of IVC. This means that any NIVC scheme is also an IVC scheme.

We begin by defining NIVC in the traditional sense in the common reference string (CRS) with preprocessing model. We consider an adaptive adversary that can pick functions $(F_1, \ldots, F_\ell)$ and φ as well as the statement after seeing the CRS.

Definition 5.5 (Non-Uniform IVC, Traditional). A *non-uniform incrementally verifiable computation* (NIVC) scheme is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and a deterministic $\mathcal{K}$ denoting the generator, the prover, the verifier, and the encoder respectively, with the following interface:

- $\mathcal{G}(1^\lambda, N) \to \mathsf{pp}$: on input security parameter λ and size bounds N, samples public parameters pp.
- $\mathcal{K}(\mathsf{pp}, ((F_1, \ldots, F_\ell), \varphi)) \to (\mathsf{pk}, \mathsf{vk})$: on input public parameters pp, a control function φ, and functions $F_1, \ldots, F_\ell$ deterministically produces a prover key pk and a verifier key vk.
- $\mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i) \to \Pi_{i+1}$: on input a prover key pk, a counter i, initial input z_0, claimed output after i applications z_i, a non-deterministic advice ω_i, and an NIVC proof Π_i attesting to z_i, produces a new proof Π_{i+1} attesting to $z_{i+1} = F_{\varphi(z_i, \omega_i)}(z_i, \omega_i)$.
- $\mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) \to \{0, 1\}$: on input a verifier key vk, a counter i, an initial input z_0, a claimed output after i applications z_i, and an NIVC proof Π_i attesting to z_i, outputs 1 if Π_i is accepting, 0 otherwise.

An NIVC scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ satisfies following requirements.

(i) Completeness: For any PPT adversary $\mathcal{A}$

$$\Pr\left[\, b = 1 \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ (\varphi, (F_1, \ldots, F_\ell), (i, z_0, z_i), (\omega_i, \Pi_i)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (\varphi, (F_1, \ldots, F_\ell))), \\ \mathcal{V}(\mathsf{vk}, (i, z_0, z_i), \Pi_i) = 1, \\ z_{i+1} \leftarrow F_{\varphi(z_i, \omega_i)}(z_i, \omega_i), \\ \Pi_{i+1} \leftarrow \mathcal{P}(\mathsf{pk}, (i, z_0, z_i), \omega_i, \Pi_i), \\ b \leftarrow \mathcal{V}(\mathsf{vk}, (i+1, z_0, z_{i+1}), \Pi_{i+1}) \end{array} \right] = 1$$

where $\ell \geq 1$ and φ produces an element in $\mathbb{Z}^*_{\ell+1}$. Moreover, φ and each F_j for $j \in \{1, \ldots, \ell\}$ are a polynomial-time computable function represented as arithmetic circuits.

(ii) Knowledge soundness: Consider constant $n \in \mathbb{N}$. For all expected polynomial-time adversaries $\mathcal{P}^*$ there exists an expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr_{\mathsf{r}}\left[\, \begin{array}{l} z_n = z \text{ where} \\ z_{i+1} \leftarrow F_{\varphi(z_i, \omega_i)}(z_i, \omega_i) \\ \forall i \in \{0, \ldots, n-1\} \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda, N), \\ (((F_1, \ldots, F_\ell), \varphi), (z_0, z), \Pi) \leftarrow \mathcal{P}^*(\mathsf{pp}, \mathsf{r}), \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, ((F_1, \ldots, F_\ell), \varphi)), \\ \mathcal{V}(\mathsf{vk}, (n, z_0, z), \Pi) = 1, \\ (\omega_0, \ldots, \omega_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}, \mathsf{r}) \end{array} \right] \approx 1$$

where r denotes an arbitrarily long random tape.

(iii) Succinctness: The NIVC proof size is independent of the iteration count.

(iv) Efficiency: The prover's time complexity at any step i is linear in the size of the function applied at step i and the total number of functions ℓ.

As with IVC, to circumvent various theoretical challenges associated with polynomial-depth recursion, we redefine NIVC as a reduction of knowledge for a single recursive step. We start by defining the *non-uniform evaluation relation* which uses a control function φ to pick from a list of functions to check the evaluation relation against as dictated by NIVC.

Definition 5.6 (Non-Uniform Evaluation Relation). We define the non-uniform evaluation relation NEval as follows

$$\mathsf{NEval} = \left\{ \begin{array}{l} ((F_1, \ldots, F_\ell), \varphi), \\ (x, y), \\ w \end{array} \;\middle|\; \begin{array}{l} x, y \in \mathsf{U}, w \in \mathsf{W}, \\ y = F_{\varphi(x,w)}(x, w) \end{array} \right\}$$

where $F_1, \ldots, F_\ell : \mathsf{U} \times \mathsf{W} \to \mathsf{U}$ and $\varphi : \mathsf{U} \times \mathsf{W} \to [\ell]$ are arithmetic circuits.

Using the non-uniform evaluation relation, we can crisply capture the correctness property of NIVC as follows.

Definition 5.7 (Non-Uniform IVC (IVC)). A non-uniform incrementally verifiable computation scheme is defined by (Proof, Π) where

(i) Proof is a ternary relation over a structure consisting of $\ell + 1$ arithmetic circuits $F_1, \ldots, F_\ell : \mathsf{U} \times \mathsf{W} \to \mathsf{U}$, and $\varphi : \mathsf{U} \times \mathsf{W} \to [\ell]$ an instance consisting of (i, z_0, z_i) where $i \in \mathbb{N}$ represents the iteration count, $z_0 \in \mathsf{U}$ represents the initial input, and $z_i \in \mathsf{U}$ represents the final output, and a fixed-size witness (referred to as the NIVC proof).

(ii) Π is a zero-interactive reduction of knowledge of type $\mathsf{NEval} \overset{\sim_1}{\times} \mathsf{Proof} \overset{\sim_2}{\longrightarrow} \mathsf{Proof}$ where $(z_i, z_{i+1}) \sim_1 (i, z_0, z_i)$ and $((z_i, z_{i+1}), (i, z_0, z_i)) \sim_2 (i + 1, z_0, z_{i+1})$ such that the prover's time complexity linear in the size of the function applied in NEval and the total number of functions ℓ in the structure.

5.2.2 Construction

Construction 5.2 (NIVC). Consider a relation $\mathcal{R}_1$ and a committed relation $\mathcal{R}_2$ for a commitment scheme $(\mathsf{Com}, \mathsf{Gen})$. Let FS be an IVC-compatible non-interactive folding scheme of type $\mathcal{R}_1 \times \mathcal{R}_2 \to \mathcal{R}_1$. Let $(\mathsf{u}_\perp, \mathsf{w}_\perp)$ be a default instance-witness pair for $\mathcal{R}_1$ that satisfies any structure and public parameters. We construct an NIVC scheme as follows.

Consider a deterministic polynomial-time function φ and ℓ polynomial-time functions $(F_1, \ldots, F_\ell)$ that take non-deterministic input and a succinct commitment scheme $(\mathsf{gen}, \mathsf{com})$. We begin by defining augmented functions F'_j for $j \in [\ell]$ as follows, where all input arguments are taken as non-deterministic advice.

$F'_j((\mathsf{pp}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{FS}}), \mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \pi) \to \mathsf{x}$:

1. Compute the next program counter $\mathsf{pc}_{i+1} \in [\ell] \leftarrow \varphi(z_i, \omega_i)$.
2. If $i = 0$:
   (a) Check that $z_0 = z_i$.
   (b) Let $\mathsf{U}_{i+1} \leftarrow (\mathsf{u}_\perp, \ldots, \mathsf{u}_\perp)$.
3. Otherwise:
   (a) Parse u_i as (C, u'_i), a commitment to the witness and the remainder.
   (b) Check that u'_i references U_i in the output of the prior iteration of F'.

$$\mathsf{u}'_i \overset{?}{=} \mathsf{enc}_{\mathsf{inst}}(\mathsf{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}_{\mathsf{FS}}, i, z_0, z_i, \mathsf{U}_i), \mathsf{pc}_i))$$

   (c) Check that $1 \leq \mathsf{pc}_i \leq \ell$.
   (d) Copy $\mathsf{U}_{i+1} \leftarrow \mathsf{U}_i$ and update $\mathsf{U}_{i+1}[\mathsf{pc}_i] \leftarrow \mathsf{FS}.\mathcal{V}(\mathsf{vk}_{\mathsf{FS}}[\mathsf{pc}_i], \mathsf{U}_i[\mathsf{pc}_i], \mathsf{u}_i, \pi)$.
   (e) Output $\mathsf{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}_{\mathsf{FS}}, i + 1, z_0, F_j(z_i, \omega_i), \mathsf{U}_{i+1}, \mathsf{pc}_{i+1}))$.

Given the augmented function F', we define the NIVC proof relation Proof as follows.

$\underline{\mathsf{Proof}((\mathsf{pp}_{\mathsf{FS}}, \mathsf{pp}_{\mathsf{com}}), ((F_1, \ldots, F_\ell), \varphi), (i, z_0, z_i), \Pi_i) \to \{0,1\}}$:

1. If $i = 0$, check that $z_i = z_0$.
2. Otherwise:
   (a) Compute $\mathsf{s}_j \leftarrow \mathsf{enc}_{\mathsf{str}}(F'_j)$ for all $j \in [\ell]$.
   (b) Compute $\mathsf{vk}_{\mathsf{FS}}[j] \leftarrow \mathsf{FS}.\mathcal{K}(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}_j)$ for all $j \in [\ell]$.
   (c) Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$
   (d) Parse u_i as (C, u'_i). Check that $\mathsf{u}'_i = \mathsf{enc}_{\mathsf{inst}}(\mathsf{com}(\mathsf{pp}_{\mathsf{com}}, (\mathsf{vk}_{\mathsf{FS}}, i, z_0, z_i, \mathsf{U}_i, \mathsf{pc}_i)))$.
   (e) Check that $1 \leq \mathsf{pc}_i \leq \ell$.
   (f) Check that $(\mathsf{pp}, \mathsf{s}_j, \mathsf{U}_i[j], \mathsf{W}_i[j]) \in \mathcal{R}_1$ and $(\mathsf{pp}, \mathsf{s}_{\mathsf{pc}_i}, \mathsf{u}_i, \mathsf{w}_i) \in \mathcal{R}_2$.

Next, we define the zero-interactive reduction $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ as follows.

$\underline{\mathcal{G}(\lambda, n) \to \mathsf{pp}}$:

1. Output $(\mathsf{pp}_{\mathsf{FS}}, \mathsf{pp}_{\mathsf{com}}) \leftarrow (\mathsf{FS}.\mathcal{G}(\lambda, n), \mathsf{gen}(\lambda, n))$.

$\underline{\mathcal{K}(\mathsf{pp}, F) \to (\mathsf{pk}, \mathsf{vk})}$:

1. Compute structures $\mathsf{s}_j \leftarrow \mathsf{enc}_{\mathsf{str}}(F'_j)$ for all $j \in [\ell]$.
2. Compute folding keys $(\mathsf{pk}_{\mathsf{FS},j}, \mathsf{vk}_{\mathsf{FS},j}) \leftarrow \mathsf{FS}.\mathcal{K}(\mathsf{pp}_{\mathsf{FS}}, \mathsf{s}_j)$ for all $j \in [\ell]$.
3. Compute the prover key
$$\mathsf{pk} \leftarrow (\mathsf{pp}, ((F_1, \ldots, F_\ell), \varphi), (\mathsf{pk}_{\mathsf{FS},1}, \ldots, \mathsf{pk}_{\mathsf{FS},\ell}), (\mathsf{vk}_{\mathsf{FS},1}, \ldots, \mathsf{vk}_{\mathsf{FS},\ell})).$$
4. Output the prover and verifier key $(\mathsf{pk}, \perp)$.

$\underline{\mathcal{P}(\mathsf{pk}, ((z_i, z_{i+1}), \omega_i) \in \mathsf{Eval}, ((i, z_0, z_i), \Pi_i) \in \mathsf{Proof}) \to ((i+1, z_0, z_{i+1}), \Pi_{i+1}) \in \mathsf{Proof}}$:

1. Parse Π_i as $((\mathsf{U}_i, \mathsf{W}_i), (\mathsf{u}_i, \mathsf{w}_i), \mathsf{pc}_i)$
2. Compute the next program counter $\mathsf{pc}_{i+1} \in [\ell] \leftarrow \varphi(z_i, \omega_i)$.
3. If $i = 0$, let
$$(\mathsf{U}_{i+1}, \mathsf{W}_{i+1}, \pi) \leftarrow ((\mathsf{u}_\perp, \ldots, \mathsf{u}_\perp), (\mathsf{w}_\perp, \ldots, \mathsf{w}_\perp), \perp)$$
Otherwise, copy $\mathsf{U}_{i+1} \leftarrow \mathsf{U}_i$ and $\mathsf{W}_{i+1} \leftarrow \mathsf{W}_i$, and update
$$(\mathsf{U}_{i+1}[\mathsf{pc}_i], \mathsf{W}_{i+1}[\mathsf{pc}_i]), \pi \leftarrow \mathsf{FS}.\mathcal{P}(\mathsf{pk}[\mathsf{pc}_i], (\mathsf{U}_i[\mathsf{pc}_i], \mathsf{W}_i[\mathsf{pc}_i]), (\mathsf{u}_i, \mathsf{w}_i)).$$
4. Compute $y \leftarrow F'_{\mathsf{pc}_{i+1}}((\mathsf{pp}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{FS}}), \mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \pi)$

5. Compute an instance-witness pair encoding the valid execution of $F'_{\mathsf{pc}_{i+1}}$.

$$(\mathsf{u}'_{i+1}, \mathsf{w}_{i+1}) \leftarrow \mathsf{enc}(F'_{\mathsf{pc}_{i+1}}, (((\mathsf{pp}_{\mathsf{com}}, \mathsf{vk}_{\mathsf{FS}}), \mathsf{U}_i, \mathsf{u}_i, \mathsf{pc}_i, (i, z_0, z_i), \omega_i, \pi), y), \omega_i).$$

6. Compute the committed instance

$$\mathsf{u}_{i+1} \leftarrow (\mathsf{u}'_{i+1}, \mathsf{Commit}(\mathsf{pp}_{\mathsf{FS}}, \mathsf{w}_{i+1})).$$

7. Let $\Pi_{i+1} \leftarrow ((\mathsf{U}_{i+1}, \mathsf{W}_{i+1}), (\mathsf{u}_{i+1}, \mathsf{w}_{i+1}), \mathsf{pc}_{i+1})$ and output $((i+1, z_0, z_{i+1}), \Pi_{i+1})$.

$\underline{\mathcal{V}(\mathsf{vk}, (z_i, z_{i+1}), (i, z_0, z_i))}$:

1. Output $(i+1, z_0, z_{i+1})$.

Theorem 5.4. Construction 5.2 is an NIVC scheme.

Proof (Intuition). Correctness follows from the same reasoning as Theorem 5.1. Kothapalli and Setty [99] provide a formal proof. □

Chapter 6

Proofs of Knowledge for NP

This chapter contains joint work with Bryan Parno [100], Srinath Setty [132], and Ioanna Tzialla [101, 132].

You might be forgiven for thinking that [the sumcheck protocol] is the only protocol in the space of zero-knowledge arguments.

– Jonathan Bootle,
Talk on Sumcheck Arguments and their Applications

In Section 1.1, we discussed how Shamir reduced the task of checking circuit satisfiability to the task of checking a sum of evaluations over a related polynomial via arithmetization, and then used the famous sumcheck protocol due to Lund et al. [106] to reduce the sumcheck statement into the task of checking a single evaluation over the related polynomial. Perhaps surprisingly, the sumcheck protocol only requires a logarithmic amount of communication and verifier complexity in the size of the polynomial. This enables a vastly limited verifier to still verify complex statements proposed by the prover. As such, the sumcheck protocol has since become a cornerstone technique in the literature.

In this section, we will develop two proofs of knowledge for NP-complete languages from prior sections using the sumcheck protocol as a fundamental building block. These proof systems can be used in conjunction with prior reductions, or in a standalone setting. First, we begin by developing a proof of knowledge for committed relaxed R1CS (Definition 4.15), which can be used in conjunction with the folding scheme for relaxed R1CS (Section 4.2) to first fold many instances and then efficiently check the final instance. Next, we present

a complementary proof of knowledge for *SIMD R1CS*, a variant of R1CS which better encodes circuits with repeating structure.

6.1 A Proof of Knowledge for Relaxed R1CS

In this section, we present a proof of knowledge for committed relaxed R1CS (Definition 4.15) by adapting Spartan [124].

6.1.1 Overview

Consider public parameters pp and size bounds m, n, and ℓ and let $s = \log m$. Without loss of generality, we assume that m and n are powers of 2 and that $m = 2 \cdot (\ell + 1)$. Consider the committed relaxed R1CS relation with respect to a polynomial commitment scheme Com. Consider a committed relaxed R1CS structure consisting of sparse matrices $(A, B, C) \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. Consider an instance $(\overline{E}, u, \overline{W}, \mathsf{x})$, and a corresponding witness (E, W, r_E, r_W).

Suppose the prover would like to demonstrate to the verifier that

$$AZ \circ BZ = u \cdot CZ + E \tag{6.1}$$

for $Z = (W, \mathsf{x}, u)$ and that $\overline{E} = \mathsf{Com}(\mathsf{pp}, E, r_E)$ and $\overline{W} = \mathsf{Com}(\mathsf{pp}, W, r_W)$.

We begin by interpreting the matrices A, B, and C as functions with the signature $\{0,1\}^{\log m} \times \{0,1\}^{\log m} \to \mathbb{F}$ in a natural manner. In particular, an input in $\{0,1\}^{\log m} \times \{0,1\}^{\log m}$ is interpreted as the binary representation of an index $(i, j) \in [m] \times [m]$, and the function outputs (i, j)th entry of the matrix. As such, let $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ denote multilinear extensions of A, B, and C interpreted as functions; In particular, they are $2 \log m$-variate sparse multilinear polynomials of size n. Similarly, we interpret E and W as functions with respective signatures $\{0,1\}^{\log m} \to \mathbb{F}$ and $\{0,1\}^{\log m - 1} \to \mathbb{F}$, and let $\widetilde{E}$ and $\widetilde{W}$ denote the multilinear extensions of E and W interpreted as functions, so they are multilinear polynomials in $\log m$ and $\log m - 1$ variables respectively. Suppose that the verifier key contains commitments to sparse polynomials $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$.

Let $Z = (W, \mathsf{x}, u)$. Similar to how we interpret matrices as functions, we interpret Z and (x, u) as functions with the following respective signatures: $\{0,1\}^s \to \mathbb{F}$ and $\{0,1\}^{s-1} \to \mathbb{F}$. Observe that the multi-linear extension $\widetilde{Z}$ of Z satisfies

$$\widetilde{Z}(X_1, \ldots, X_s) = (1 - X_1) \cdot \widetilde{W}(X_2, \ldots, X_s) + X_1 \cdot \widetilde{(\mathsf{x}, u)}(X_2, \ldots, X_s) \tag{6.2}$$

Then, for

$$F(x) = \left(\sum_{y \in \{0,1\}^s} \widetilde{A}(x, y) \cdot \widetilde{Z}(y) \right) \cdot \left(\sum_{y \in \{0,1\}^s} \widetilde{B}(x, y) \cdot \widetilde{Z}(y) \right) - \tag{6.3}$$

$$\left(u \cdot \sum_{y \in \{0,1\}^s} \widetilde{C}(x, y) \cdot \widetilde{Z}(y) + \widetilde{E}(x) \right), \tag{6.4}$$

checking Equation 6.1 is equivalent to checking that

$$0 = F(x) \quad \text{for all } x \in \{0,1\}^s. \tag{6.5}$$

We now want to re-express Equation (6.5) as a single sumcheck statement rather than multiple evaluation checks. To do so, we observe that we can arrange the evaluations of F on $\{0,1\}^s$ on a new polynomial G and have these evaluations scale the Lagrange polynomials for the space $\{0,1\}^s$ In particular, checking Equation 6.5 is equivalent to checking that all of the coefficients of the polynomial

$$G(X) = \sum_{x \in \{0,1\}^s} \widetilde{\mathsf{eq}}(X, x) \cdot F(x) \tag{6.6}$$

are zero. In other words, we must check that the above polynomial is the zero polynomial. Recall that $\widetilde{\mathsf{eq}}(X, x) = 1$ if $X = x$ and 0 otherwise for $X, x \in \{0,1\}^s$. By the Schwartz-Zippel Lemma (Lemma 4.6), the verifier can check Equation 6.6 with overwhelming probability by sampling $\tau \xleftarrow{\$} \mathbb{F}$ and checking that

$$0 = \sum_{x \in \{0,1\}^s} \widetilde{\mathsf{eq}}(\tau, x) \cdot F(x). \tag{6.7}$$

To compute the right-hand side in Equation (6.7), the prover and verifier recursively apply the sumcheck protocol (Construction 4.3) to reduce to the task of checking

$$\sigma = \widetilde{\mathsf{eq}}(\tau, r_x) \cdot F(r_x). \tag{6.8}$$

for some random point $r_x \in \mathbb{F}^s$ and resulting evaluation σ.

Now, the verifier can locally evaluate $\widetilde{\mathsf{eq}}(\tau, r_x)$ in $O(\log m)$ field operations by computing

$$\widetilde{\mathsf{eq}}(\tau, r_x) = \prod_{i=1}^{s} \left(\tau_i \cdot r_{x,i} + (1 - \tau_i) \cdot (1 - r_{x,i})\right).$$

The prover helps the verifier compute the remainder of $G(r_x)$ by computing and sending

$$\sigma_A = \sum_{y \in \{0,1\}^s} \widetilde{A}(r_x, y) \cdot \widetilde{Z}(y) \tag{6.9}$$

$$\sigma_B = \sum_{y \in \{0,1\}^s} \widetilde{B}(r_x, y) \cdot \widetilde{Z}(y) \tag{6.10}$$

$$\sigma_C = \sum_{y \in \{0,1\}^s} \widetilde{C}(r_x, y) \cdot \widetilde{Z}(y) \tag{6.11}$$

$$e = \widetilde{E}(r_x) \tag{6.12}$$

The verifier indeed checks that

$$\widetilde{eq}(\tau, r_x) \cdot (\sigma_A \cdot \sigma_B - u \cdot \sigma_C + e),$$

however, the verifier must still check the claimed terms sent by the prover. The last term can be efficiently checked with a single polynomial evaluation query to $\widetilde{E}$ using a polynomial evaluation proof. The first three terms can be checked by applying the sumcheck protocol three more times in parallel over the variable y, once to each of the following three polynomials using the same randomness $r_y \in \mathbb{F}^s$ in each of the three invocations. Once again, this reduces the task of checking the prior equations to the task of checking

$$\sigma'_A = \widetilde{A}(r_x, r_y) \cdot \widetilde{Z}(r_y) \tag{6.13}$$

$$\sigma'_B = \widetilde{B}(r_x, r_y) \cdot \widetilde{Z}(r_y) \tag{6.14}$$

$$\sigma'_C = \widetilde{C}(r_x, r_y) \cdot \widetilde{Z}(r_y) \tag{6.15}$$

for some σ'_A, σ'_B, and σ'_C produced during the protocol.

At this point, the verifier can use the sparse polynomial commitment scheme to verifiably (and efficiently) evaluate $\widetilde{A}(r_x, r_y)$, $\widetilde{B}(r_x, r_y)$, and $\widetilde{C}(r_x, r_y)$. Moreover, by Equation (6.2) the verifier can evaluate $\widetilde{Z}(r_y)$ by (locally) computing $\widetilde{(\mathsf{x}, u)}(r_y)$ and verifiably querying $\widetilde{W}(r_y)$.

6.1.2 Construction

We formally present a proof of knowledge for committed relaxed R1CS below.

Construction 6.1 (Spartan for Relaxed R1CS). Let $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit}, \Pi_{\mathsf{PC}})$ denote a multilinear polynomial commitment scheme (Definition 4.10) and let $\mathsf{PCSparse} = (\mathsf{GenSparse}, \mathsf{CommitSparse}, \Pi_{\mathsf{PCSparse}})$ denote a multilinear polynomial commitment scheme for sparse polynomials. We construct a proof of knowledge for committed relaxed R1CS (Definition 4.15) with respect to commitment scheme PC. We define the generator, encoder, prover and verifier as follows.

$\underline{\mathcal{G}(1^\lambda, (m, n, \ell)) \to \mathsf{pp}}$:

1. Compute $\mathsf{pp}_{\mathsf{PC}} \leftarrow \mathsf{Gen}(m)$ and $\mathsf{pp}_{\mathsf{PCSparse}} \leftarrow \mathsf{GenSparse}(m)$
2. Output $\mathsf{pp} \leftarrow ((\mathsf{pp}_{\mathsf{PC}}, \mathsf{pp}_{\mathsf{PCSparse}}), (m, n, \ell))$.

$\underline{\mathcal{K}(\mathsf{pp}, (A, B, C)) \to (\mathsf{pk}, \mathsf{vk})}$:

1. Let $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ be the multilinear extensions of A, B, and C.
2. Compute commitments

$$\overline{A} \leftarrow \mathsf{CommitSparse}(\mathsf{pp}_{\mathsf{PCSparse}}, \widetilde{A}),$$
$$\overline{B} \leftarrow \mathsf{CommitSparse}(\mathsf{pp}_{\mathsf{PCSparse}}, \widetilde{B}),$$
$$\overline{C} \leftarrow \mathsf{CommitSparse}(\mathsf{pp}_{\mathsf{PCSparse}}, \widetilde{C})$$

3. Output $\mathsf{pk} \leftarrow (\widetilde{A}, \widetilde{B}, \widetilde{C})$ and $\mathsf{vk} \leftarrow (\overline{A}, \overline{B}, \overline{C})$.

$\underline{\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), (\overline{E}, u, \overline{W}, \mathsf{x}), (E, W, r_E, r_W))}$:

1. $\mathcal{V} \to \mathcal{P}$: Let $s \leftarrow \log m$. $\mathcal{V}$ samples $\tau \xleftarrow{\$} \mathbb{F}^s$ and sends it to $\mathcal{P}$.

2. $\mathcal{V} \leftrightarrow \mathcal{P}$: Let the polynomial F be defined as in Equation (6.3). Let the commitment $\overline{F}$ to F be defined as $((\overline{A}, \overline{B}, \overline{C}, \overline{E}, u, \overline{W}, \mathsf{x}), 0, ())$. Recursively run the sumcheck protocol (Construction 4.3) on input instance $(\overline{F}, 0, ())$ and witness F where F is defined as in Equation (6.3). Let the reduced polynomial evaluation instance be $(\overline{F}, \sigma, r_x) \in (\mathbb{F}, \mathbb{F}^s)$.

3. $\mathcal{P} \to \mathcal{V}$: Compute and send the following terms to the verifier

$$\sigma_A \leftarrow \sum_{y \in \{0,1\}^s} \widetilde{A}(r_x, y) \cdot \widetilde{Z}(y) \tag{6.16}$$

$$\sigma_B \leftarrow \sum_{y \in \{0,1\}^s} \widetilde{B}(r_x, y) \cdot \widetilde{Z}(y) \tag{6.17}$$

$$\sigma_C \leftarrow \sum_{y \in \{0,1\}^s} \widetilde{C}(r_x, y) \cdot \widetilde{Z}(y) \tag{6.18}$$

$$e \leftarrow \widetilde{E}(r_x) \tag{6.19}$$

4. $\mathcal{V}$: Check that

$$\widetilde{eq}(\tau, r_x) \cdot (\sigma_A \cdot \sigma_B - u \cdot \sigma_C + e),$$

5. $\mathcal{V} \leftrightarrow \mathcal{P}$: Check that $e = \widetilde{E}(r_x)$ using the polynomial evaluation proof Π_{PC}.

6. $\mathcal{V} \leftrightarrow \mathcal{P}$: For $M \in \{A, B, C\}$ let

$$F_M(Y) = \widetilde{M}(r_x, Y) \cdot \widetilde{Z}(Y)$$

where $\widetilde{Z}$ is defined as in Equation (6.2). Let the corresponding commitment $\overline{F}_M$ to F_M be defined as $(\overline{M}, \overline{W}, u, \mathsf{x})$. For each $M \in \{A, B, C\}$, recursively run the sumcheck protocol on input instance $(\overline{F}_M, \sigma_A, ())$ and witness F_M on the same verifier randomness r_y. Let the reduced polynomial evaluation instances be $(\overline{F}_M, \sigma'_M, r_y)$.

7. $\mathcal{P}$: Compute and send $v_M \leftarrow \widetilde{M}(r_x, r_y)$ for $M \in \{A, B, C\}$ and $v_Z \leftarrow \widetilde{Z}(r_y)$.

8. $\mathcal{V}$: Check that $\sigma_M = v_M \cdot v_Z$ for all $M \in \{A, B, C\}$.

9. $\mathcal{V} \leftrightarrow \mathcal{P}$: For $M \in \{A, B, C\}$, check that $v_M = \widetilde{M}(r_x, r_y)$ using the polynomial evaluation proof system Π_{PCSparse}.

10. $\mathcal{V} \leftrightarrow \mathcal{P}$: Check that $v_Z = \widetilde{Z}(r_y)$ by using the polynomial evaluation proof system Π_{PC} to compute the evaluation $\widetilde{W}(r_y)$ and computing $\widetilde{Z}(r_y)$ according to Equation (6.2).

Theorem 6.1 (Spartan for Committed Relaxed R1CS). Construction 6.1 is a proof of knowledge for committed relaxed R1CS defined with respect to a polynomial commitment scheme, defined over a finite field $\mathbb{F}$, with the following parameters, where m denotes the dimension of the R1CS matrices, and n denotes the number of non-zero entries in the matrices.

- $O(\log m)$ round complexity
- $O(\log m) \cdot |\mathbb{F}| + 3 \cdot |\Pi_{\mathsf{PCSparse}}| + 2 \cdot |\Pi_{\mathsf{PC}}|$ communication complexity and verifier time complexity.
- $O(n) \cdot |\mathbb{F}| + 3 \cdot |\Pi_{\mathsf{PCSparse}}| + 2 \cdot |\Pi_{\mathsf{PC}}|$ prover time complexity.

Proof. Perfect completeness follows from perfect completeness of the sum-check protocol and the fact that Equation (6.7) holds with probability 1 over the choice of τ if φ is satisfiable.

The sum-check protocol is applied four times. In each invocation, the polynomial to which the sum-check protocol is applied has degree at most 3 in each variable, and the number of variables is $s = \log m$. Hence, the round complexity of the polynomial IOP is $O(\log m)$. Since each polynomial has degree at most 3 in each variable, the total communication cost is $O(\log m)$ field elements.

The claimed verifier runtime is immediate from the verifier's runtime in the sum-check protocol, and the fact that $\widetilde{\mathsf{eq}}$ can be evaluated at any input $(\tau, r_x) \in \mathbb{F}^{2s}$ in $O(\log m)$ field operations. As in Spartan [124], the prover's work is $O(n)$ operations over $\mathbb{F}$ using prior techniques [59, 137]. □

6.1.3 Instantiating the Polynomial Commitment Schemes

We now discuss options for instantiating the polynomial commitment schemes Π_{PC} and Π_{PCSparse} from Construction 6.1.

Polynomial Commitments for Multilinear Polynomials We can naturally interpret commitments to m-sized vectors over $\mathbb{F}$ are commitments to $\log m$-variate multilinear polynomials represented with evaluations over $\{0,1\}^m$ [102, 124, 136, 141]. Furthermore, we can naturally derive a polynomial commitment scheme for $\log m$-variate multilinear polynomials given a proof of knowledge to prove an inner product computation between a committed vector and an m-sized public vector $((r_1, 1-r_1) \otimes \ldots \otimes (r_{\log m}, 1-r_{\log m}))$, where $r \in \mathbb{F}^{\log m}$ is an evaluation point. There are two candidate constructions in the literature, which differ primarily in the verifier's time complexity.

1. $\mathsf{PC}_{\mathsf{BP}}$: If the commitment scheme for vectors over $\mathbb{F}$ is Pedersen commitments, as in prior work [136], Bulletproofs [42] provides a suitable inner-product proof protocol. The polynomial commitment scheme here achieves the following efficiency characteristics, assuming the hardness of the discrete logarithm problem. For a $\log m$-variate multilinear polynomial, committing takes $O_\lambda(m)$ time to produce an $O_\lambda(1)$-sized

commitment; the prover incurs $O_\lambda(m)$ costs to produce an evaluation proof of size $O_\lambda(\log m)$ that can be verified in $O_\lambda(m)$. Note that $\mathsf{PC}_{\mathsf{BP}}$ is a special case of Hyrax's polynomial commitment scheme [136].

2. $\mathsf{PC}_{\mathsf{Dory}}$: If vectors over $\mathbb{F}$ are committed with a two-tiered "matrix" commitment (see for example, [46, 102]), which provides $O_\lambda(1)$-sized commitments to m-sized vectors under the SXDH assumption. With this commitment scheme, Dory [102] provides the necessary inner-product proof. The polynomial commitment here achieves the following efficiency characteristics, assuming the hardness of SXDH. For a $\log m$-variate multilinear polynomial, committing takes $O_\lambda(m)$ time to produce an $O_\lambda(1)$-sized commitment; the prover incurs $O_\lambda(m)$ costs to produce an evaluation proof of size $O_\lambda(\log m)$ that can be verified in $O_\lambda(\log m)$.

Polynomial commitments for sparse multilinear polynomials. In our construction, we require polynomial commitment schemes that can *efficiently* handle sparse multilinear polynomials. Spartan [124] provides a generic compiler to transform existing polynomial commitment schemes for multilinear polynomials into those that can efficiently handle sparse multilinear polynomials. Specifically, we apply [103, Theorem 5], which captures Spartan's compiler in a generic manner, to $\mathsf{PC}_{\mathsf{BP}}$ and $\mathsf{PC}_{\mathsf{Dory}}$ to obtain their variants that can efficiently handle sparse multilinear polynomials; we refer to them as Sparse-$\mathsf{PC}_{\mathsf{BP}}$ and Sparse-$\mathsf{PC}_{\mathsf{Dory}}$ respectively.

Theorem 6.2 (Spartan for Committed Relaxed R1CS from $\mathsf{PC}_{\mathsf{BP}}$). Assuming the hardness of the discrete logarithm problem, instantiating Construction 6.1 with $\mathsf{PC}_{\mathsf{BP}}$ and Sparse-$\mathsf{PC}_{\mathsf{BP}}$ results in a proof of knowledge for committed relaxed R1CS with the following efficiency characteristics, where m denotes the dimensions of R1CS matrices and n denotes the number of non-zero entries in the matrices: The encoder runs in time $O_\lambda(n)$; The prover runs in time $O_\lambda(n)$; The proof length is $O_\lambda(\log n)$; and the verifier runs in time $O_\lambda(n)$.

Proof. Using Sparse-$\mathsf{PC}_{\mathsf{BP}}$, the encoder takes $O_\lambda(n)$ time to create commitments $2\log m$-variate sparse multilinear polynomials $\widetilde{A}, \widetilde{B}, \widetilde{C}$. The prover's costs in the proof of knowledge is $O(n)$. Furthermore, proving the evaluations of two $O(\log m)$-variate multilinear polynomials using $\mathsf{PC}_{\mathsf{BP}}$ takes $O_\lambda(m)$ time. To prove the evaluations of three $2\log m$-variate sparse multilinear polynomials of size n using Sparse-$\mathsf{PC}_{\mathsf{BP}}$ takes $O_\lambda(n)$ time. In total, the prover time is $O_\lambda(n)$. The proof length in the proof of knowledge is $O(\log m)$, and the proof sizes in the polynomial evaluation proofs is $O_\lambda(\log n)$. Therefore, the total proof length is $O_\lambda(\log n)$. The verifier's time in the proof of knowledge is $O(\log m)$. In addition, it verifies five polynomial evaluations, which costs $O_\lambda(n)$ time: the two polynomial in the instance take $O_\lambda(m)$ time using $\mathsf{PC}_{\mathsf{BP}}$, and the three polynomials in the structure takes $O_\lambda(n)$ time using Sparse-$\mathsf{PC}_{\mathsf{BP}}$. Therefore, in total, the verifier time is $O_\lambda(n)$. □

Corollary 6.1 (Spartan for Committed Relaxed R1CS from $\mathsf{PC}_{\mathsf{Dory}}$). Assuming the hardness of the SXDH problem, instantiating Construction 6.1 with $\mathsf{PC}_{\mathsf{Dory}}$ and Sparse-$\mathsf{PC}_{\mathsf{Dory}}$ results in a proof of knowledge for committed relaxed R1CS with the following

efficiency characteristics, where m denotes the dimensions of R1CS matrices and n denotes the number of non-zero entries in the matrices: The encoder runs in time $O_\lambda(n)$; The prover runs in time $O_\lambda(n)$; The proof length is $O_\lambda(\log n)$; and the verifier runs in time $O_\lambda(\log n)$.

6.2 A Proof of Knowledge for SIMD R1CS

In practice, we are often concerned with proving the execution of circuits with a repeated subcircuit. We refer to such circuits as SIMD (i.e., same-instruction multiple-data) circuits. We can equivalently define SIMD R1CS, which considers the R1CS (Definition 4.13) encoding of such circuits. In this section, we are concerned with developing a proof of knowledge for SIMD R1CS which leverages the structured nature of the computation to achieve a more efficient proof system.

Recall that in Spartan for relaxed R1CS (Construction 6.1), the prover and verifier interactively reduce the task of checking the original R1CS instance $((A, B, C), (\overline{E}, u, \overline{W}, \mathsf{x}))$ into the task of evaluating a random point on the corresponding multilinear extensions $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$, $\widetilde{E}$, and $\widetilde{W}$. If we naively applied Spartan to the SIMD circuit, the matrices A, B, and C would scale with the size of the entire circuit, as would the polynomial evaluation queries to the corresponding multilinear extension. Instead, we redesign the statement to ensure that the matrices A, B, and C only scale with a single subcircuit, and that the verifier's queries to these matrices are amortized over all repeated subcircuit evaluations. We refer to this variant of Spartan as SIMD Spartan. The techniques presented in this section are orthogonal and complementary to that of Section 6.1 which modifies Spartan to handle committed relaxed R1CS. Combining the two sets of techniques, we can naturally derive a proof of knowledge for a SIMD variant of relaxed R1CS [132].

We begin by formally defining SIMD R1CS.

Definition 6.1 (SIMD R1CS). Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$ and commitment parameters pp. The Data-parallel R1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. An instance consists of public IO vectors $\mathsf{x} = (\mathsf{x}_1, \ldots, \mathsf{x}_\beta) \in \mathbb{F}^{\ell \times \beta}$ and a commitment $\overline{W}$. A witness $W = (W_1, \ldots, W_\beta) \in \mathbb{F}^{m-\ell-1 \times \beta}$ is satisfying if

$$(A \cdot Z) \circ (B \cdot Z) = C \cdot Z,$$

where $Z = (W, \mathsf{x}, \mathbf{1})$ and $\overline{W} \leftarrow \mathsf{Commit}(\mathsf{pp}, W)$.

To check a SIMD R1CS instance using known algebraic techniques, we must first encode the instance as a set of low degree polynomials: To do so, we first interpret matrices and vectors as functions and then take their low-degree extensions (i.e. multilinear polynomials that behave like the original function on a specified domain). Specifically, let Z represent the vector that results from concatenating each Z_i for $i \in [\beta]$, and let $\widetilde{Z}$ denote the corresponding low-degree extension. Similarly let $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$ represent the low-degree extensions

of matrices A, B, C. Given these extensions, we define polynomial F which computes the satisfiability of each constraint:

$$F(k,x) = \left(\sum_{y \in \{0,1\}^{\log m}} \widetilde{A}(x,y) \cdot \widetilde{Z}(k,y)\right) \cdot \left(\sum_{y \in \{0,1\}^{\log m}} \widetilde{B}(x,y) \cdot \widetilde{Z}(k,y)\right) - \left(\sum_{y \in \{0,1\}^{\log m}} \widetilde{C}(x,y) \cdot \widetilde{Z}(k,y)\right). \tag{6.20}$$

In particular, we observe that if indeed Z is a satisfying witness, then $F(k,x) = 0$ for all $k \in \{0,1\}^{\log \beta}$ and $x \in \{0,1\}^{\log m}$.

It is still unclear how the verifier can check this property succinctly, so we instead define a *multilinear* polynomial Q: As in Section 6.1, we can re-express the task of checking many evaluations of $F(k,x)$, to the task of checking a single sumcheck by encoding the evaluations of F as the coefficients of a new polynomial G.

$$G(t_1,t_2) = \sum_{k \in \{0,1\}^{\log \beta}, x \in \{0,1\}^{\log m}} \widetilde{\mathsf{eq}}(k,t_1) \cdot \widetilde{\mathsf{eq}}(x,t_2) \cdot F(k,x)$$

where $\widetilde{\mathsf{eq}}(x,y)$ returns 1 if $x = y$ and 0 otherwise over a specified domain.

By the same reasoning as Section 6.1 if Z is a satisfying witness, we have that G must be the zero polynomial. Therefore, it is sufficient for the verifier to check that $G(\tau_1,\tau_2) = 0$ for randomly sampled $\tau_1, \tau_2 \xleftarrow{\$} \mathbb{F}$.

Now the proof of knowledge can proceed using Spartan's algebraic techniques as follows:

1. The verifier sends random $(\tau_1,\tau_2) \xleftarrow{\$} \mathbb{F}^{\log \beta} \times \mathbb{F}^{\log m}$.

2. The verifier picks random (r_k, r_x) and uses the sumcheck protocol to reduce the task of checking $G(\tau_1,\tau_2) = 0$ to the task of checking $F(r_k,r_x) = e$ for some e produced in the protocol.

3. The verifier picks random r_y and uses the sumcheck protocol to reduce the task of checking $F(r_k,r_x) = e$ to the task of checking claimed evaluations of $\widetilde{A}, \widetilde{B}, \widetilde{C}$ at (r_x,r_y) and $\widetilde{Z}(r_k,r_y)$. The first three are evaluated by the verifier locally (or by using sparse polynomial evaluation proofs [124]).

4. To enable the verifier to evaluate $\widetilde{Z}$ succinctly, we make use of polynomial evaluation proofs [92, 136]. Recall that we define $Z = (Z_1, Z_2, \ldots, Z_\beta)$ where $Z_k = (W_k, \mathsf{x}_k, 1)$. Assume that for each $k \in [\beta]$, $|W_k| = |\mathsf{x}_k| + 1 = m/2$. Observe that for all $k \in [\beta]$ and $y \in [m]$,

$$Z[k][y] = \begin{cases} W[k][y], \text{ if } y \leq m/2 \\ (\mathsf{x},1)[k][y], \text{ otherwise} \end{cases} \tag{6.21}$$

Thus, the most significant bit of y determines whether Z should evaluate W or $(\mathsf{x}, 1)$. Thus, letting the first bit be the most significant bit, we have that

$$\begin{aligned}\widetilde{Z}(r_k, r_{(y,1)}, \ldots, r_{(y,\log m)}) =& (1 - r_{(y,1)}) \cdot \widetilde{W}(r_k, r_{(y,2)}, \ldots, r_{(y,\log m)}) + \\ & r_{(y,1)} \cdot \widetilde{\mathsf{x}}(r_k, r_{(y,2)}, \ldots, r_{(y,\log m)})\end{aligned}$$

Then, the verifier can evaluate $\widetilde{Z}$ efficiently by locally evaluating

$$\widetilde{(\mathsf{x}, 1)}(r_k, r_{(y,2)}, \ldots, r_{(y,\log m)})$$

and then using a polynomial evaluation proof to evaluate $\widetilde{W}$ on the same point.

Observe that the verifier only has to participate in $\log \beta$ additional rounds, each with a constant number of field elements communicated. In exchange, the verifier only has to evaluate $\widetilde{A}$, $\widetilde{B}$, and $\widetilde{C}$ once.

Chapter 7

Transformations over Reductions of Knowledge

This chapter contains joint work with Srinath Setty and Leah Rosenbloom.

Everything provable is provable in zero-knowledge.

– Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway

As discussed in Section 1.1, cryptographers often design protocols in an idealized setting, such as one that assumes the existence of a random oracle or assumes that interaction is permissible. Such protocols are then generically transformed into protocols that satisfy stronger, or more stringent requirements, such as non-interactivity. In this chapter, we design three such transformations. First, we design a non-interactive transformation based on that of Fiat and Shamir [67]. Second, we design a *straight-line* transformation that drastically reduces the knowledge soundness error of a reduction of knowledge, and as a

result enables a polynomial number of compositions (as opposed to a constant). Finally, we design a zero-knowledge transformation, which takes any reduction of knowledge and randomizes the initial input (using a folding scheme) to achieve a zero-knowledge reduction of knowledge.

7.1 A Non-Interactive Transformation

We now present the Fiat-Shamir transformation [67] for reductions of knowledge, which transforms any interactive public-coin reduction into a non-interactive reduction in the random oracle model (thus extending all of the above techniques for interactive public-coin reductions). Recall that in a public-coin reduction the verifier only sends uniformly random challenges in response to the prover's messages (Definition 2.3). The Fiat-Shamir transformation works by simulating the verifier's responses by querying the random-oracle with the prover's messages, thereby eliminating the need for interaction.

Construction 7.1 (Fiat-Shamir Transformation). Let ρ denote a random oracle. Let $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ be a public-coin reduction of knowledge with ℓ rounds. We construct a non-interactive reduction of knowledge $\Pi' = (\mathcal{G}, \mathcal{K}', \mathcal{P}', \mathcal{V}')$ in the random oracle model as follows.

$\underline{\mathcal{K}'(\mathsf{pp}, \mathsf{s}) \to (\mathsf{pk}, \mathsf{vk})}$:

1. Compute and output $((\mathsf{pk}, \mathsf{vk}), \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$.

$\underline{\mathcal{P}'(\mathsf{pk}, u_1, w_1) \to (u_2, w_2)}$:

1. Run $\mathcal{P}(\mathsf{pk}, u_1, w_1)$. On the ith message m_i, respond with verifier randomness $r_{i+1} = \rho(m_i, r_i)$ where $r_1 = \rho(\mathsf{vk}, u_1)$. Let (u_2, w_2) be the output of $\mathcal{P}$ and let $\pi = (m_1, \ldots, m_\ell)$.
2. Send π to the verifier.
3. Output (u_2, w_2).

$\underline{\mathcal{V}'(\mathsf{vk}, u_1) \to u_2}$:

1. Receive $\pi = (m_1, \ldots, m_\ell)$ from the prover. Compute $r_{i+1} \leftarrow \rho(m_i, r_i)$ where $r_1 = \rho(\mathsf{vk}, u_1)$.
2. Run $\mathcal{V}(\mathsf{vk}, u_1)$ with randomness $(r_1, \ldots, r_\ell)$. In round i send prover message m_i. Let u_2 be the output of $\mathcal{V}$.
3. Output u_2.

Lemma 7.1 (Fiat-Shamir Transformation [67]). Construction 7.1 is a transformation from a public-coin reduction of knowledge $\Pi : \mathcal{R}_1 \xrightarrow{\sim} \mathcal{R}_2$ to a non-interactive reduction of knowledge $\Pi : \mathcal{R}_1 \xrightarrow{\sim} \mathcal{R}_2$ in the random oracle model.

7.2 A Straight-Line Transformation

One caveat we have glossed over thus far is that the sequential composition result (Theorem 2.1) abstracts away the fact that the extractors have to run extractors. This means that with each composition, the extractor runtime blows up polynomially, meaning that the overall extractor runtime is exponential in the total number of compositions. This is acceptable if we only have a constant number of compositions, such as when we are formalizing steps in a proof system for NP such as Spartan [124]. In general, however, this explains the limitations for many modern techniques today. For instance, with IVC, no existing construction can recurse indefinitely (in standard models) due to this composition bound.

As we discuss in Section 1.1, there exists a stricter notion of knowledge-soundness, known as *straight-line extractability* that ensures that the composed extractor only has an additive overhead with each composition. In particular, a reduction of knowledge in the random oracle model is straight-line extractable if an extractor can extract a witness from a single accepting transcript and the corresponding list of query-response pairs to the random oracle. Straight-line extractability means that the traditional knowledge-soundness extractor can run the prover *once* (hence the name) and then invoke the straight-line extractor on the corresponding transcript and oracle queries. As we demonstrate, this ensures that a composed extractor for two reductions composed sequentially only has to run the corresponding extractor each reduction independently *once*, maintaining only an additive runtime overhead.

Straight-line extractability ensures that we can compose reductions a polynomial number of times (as opposed to constant). Our goal in this section is to demonstrate that we can take any reduction of knowledge (with mild qualifications) and transform it into a new reduction of knowledge that satisfies this stronger straight-line property. There are standard techniques in the literature for how to achieve this (most notably Fischlin's [69] transformation), but we are faced with new challenges and opportunities in the context of composition. In particular, we must prove the closure of straight-line extractability That is, given two straight-line reductions, we must demonstrate that composing them results in a straight-line extractable reduction.

7.2.1 Overview

Our starting point for building a straight-line transformation, is the compiler of Ganesh et al. [75], which observes that any proof of knowledge can be made straight-line using a straight-line polynomial commitment to the witness. Recall from Definition 4.10 that a polynomial commitment is a commitment to a polynomial such that the commitment can later be verifiably opened at any particular evaluation point. A straight-line extractable polynomial commitment scheme ensures that that an accepting opening proof reveals the witness in the transcript and oracle queries with overwhelming probability. Ganesh et al. achieve a straight-line extractable polynomial commitment scheme by applying the Fischlin transformation [69] to a standard polynomial commitment scheme. We generalize and extend this approach for reductions of knowledge.

Our transformation is comprised of two smaller transformations. The first transformation converts a standard polynomial commitment scheme with *computationally injective proofs* (i.e., it is difficult to discover two different proofs for the same instance) into a straight-line proof of opening (Construction 7.4). Ganesh et al. [75] show that the popular KZG polynomial commitment scheme [92] provides computationally injective proofs.

The second transformation takes any straight-line proof of opening and a reduction of knowledge with commitments to the witness in the instance and converts it into a straight-line reduction of knowledge (Construction 7.5).

Construction 7.2 (Straight-Line Opening Transformation, Informal). Suppose we are provided a polynomial commitment scheme com (Definition 4.10) Our goal is to design a straight-line non-interactive proof of knowledge in the random-oracle model for the opening relation defined as follows.

$$\mathsf{Open}_{\mathsf{com}} = \{(\overline{w}, w) \mid \overline{w} = \mathsf{com}(w)\}\,.$$

We design a straight-line proof of knowledge for the opening relation as follows. Suppose the prover and verifier are provided instance $\overline{w}$ and the prover is additionally provided a corresponding opening witness w. The prover repeatedly samples evaluation points x and queries $h \leftarrow \rho(\overline{w}, x, y, \pi)$, where ρ denotes the random oracle, y is the result of evaluating w (interpreted as a polynomial) on x, and π is a proof that $w(x) = y$ generated using Π_{poly}. The prover samples until it finds $h \leq 2^b$ for hardness parameter b. The prover then sends (x, y, π) to the verifier.

The verifier checks that π indeed attests that the polynomial underlying $\overline{w}$, when evaluated at x, results in y and additionally checks that $\rho(\overline{w}, x, y, \pi) \leq 2^b$.

Lemma 7.2 (Straight-Line Opening Transformation, Informal). Construction 7.2 takes a non-interactive proof of knowledge for $\mathsf{Poly}_{\mathsf{com}}$ with computationally injective proofs (Definition 7.2) and produces a straight-line non-interactive proof of knowledge for $\mathsf{Open}_{\mathsf{com}}$ in the random-oracle model.

Proof (Sketch). For a small enough value of b, the prover will have to query a large number of evaluations of w. By setting b appropriately, we can ensure that the number of queries is greater than the degree of w with overwhelming probability. Then, given the list of query-response pairs, the extractor can interpolate these values to recover the original polynomial w.

A malicious prover may attempt flood the queries with arbitrary values of (x, y), which do not lie on the same polynomial. To prevent this, we require that the prover queries the random oracle with the proof π as well. This ensures that the extractor can pick out only the evaluation points with valid proofs. A malicious prover may also attempt to retrieve many values of h for a single value for (x, y) by only rerandomizing the proof π, which would make it impossible for the extractor to interpolate. However, this would be prohibitively expensive due to the computationally injective proofs property, which makes it difficult to find two proofs π and π' for the same value of (x, y). □

Construction 7.3 (Straight-Line Transformation, Informal). Suppose we are provided a commitment scheme com with a straight-line, non-interactive proof of knowledge Π_{open} for $\mathsf{Open}_{\mathsf{com}}$ in the random oracle model. Consider a reduction of knowledge Π of type $\mathcal{R}_{\mathsf{com}} \to \mathcal{S}$ where $\mathcal{S}$ is a relation and, for relation $\mathcal{R}$, $\mathcal{R}_{\mathsf{com}}$ denotes the relation

$$\{((u, \overline{w}), w) \mid (u, w) \in \mathcal{R}, \overline{w} = \mathsf{com}(w)\}.$$

We design a straight-line reduction of knowledge with the same type. Indeed, suppose the prover and verifier are provided instance $(u, \overline{w})$ and the prover is additionally provided a corresponding witness w. The prover first proves that it knows an opening to $\overline{w}$ by running Π_{open} with the verifier on the instance-witness pair $(\overline{w}, w)$. Next, the prover and verifier run the original reduction Π on the original instance-witness pair.

Theorem 7.1 (Straight-Line Transformation, Informal). Construction 7.3 takes a straight-line, non-interactive proof of knowledge for $\mathsf{Open}_{\mathsf{com}}$ in the random-oracle model and a reduction of knowledge of type $\mathcal{R}_{\mathsf{com}} \to \mathcal{S}$ in the random-oracle model and produces a straight-line reduction of knowledge of type $\mathcal{R}_{\mathsf{com}} \to \mathcal{S}$ in the random-oracle model.

Proof (Sketch). By the straight-line extractability property of Π_{open}, the extractor can use the transcript and queries for the overall reduction to extract an opening w to $\overline{w}$. For the sake of argument, by the knowledge soundness of Π, the extractor can also extract some w' such that $(u, w') \in \mathcal{R}$ and $\overline{w} = \mathsf{com}(w')$ (albeit not in a straight-line manner). Then, by the binding property of the commitment scheme we must have that $w' = w$. Therefore, we have that the original witness w is such that $((u, \overline{w}), w) \in \mathcal{R}_{\mathsf{com}}$. □

7.2.2 Defining Straight-Line Extractability

We now formally define *straight-line extractability*, which characterizes reductions where the extractor is able to extract an input witness given a single transcript and random-oracle query-response pairs (as defined by Fischlin [69]). Extractors restricted in this way can be composed to produce a new straight-line extractor.

Definition 7.1 (Straight-Line Extractable). A (structured) reduction of knowledge $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is straight-line extractable in the random oracle model if there exists deterministic polynomial-time extractor $\mathcal{E}$, such that for any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$, given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$, $(\mathsf{s}, u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$ and $(u_2, w_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})$ with corresponding transcript tr, and the prover's query-response pairs qr to the random oracle ρ we have that

$$\Pr[(\mathsf{pp}, \mathsf{s}, u_1, \mathcal{E}(\mathsf{pp}, u_1, (\mathsf{tr}, \mathsf{qr}, w_2))) \in \mathcal{R}_1] \approx \Pr[(\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2].$$

Our transformation for producing straight-line reductions of knowledge requires a non-interactive polynomial commitment scheme with computationally injective proofs.

Definition 7.2 (Computationally Injective Proofs). A non-interactive (structured) reduction of knowledge $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ has computationally injective proofs if for any expected-polynomial-time adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$, $(\mathsf{s}, u, \pi_1, \pi_2) \leftarrow \mathcal{A}(\mathsf{pp})$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$ we have that

$$\Pr[\mathcal{V}(\mathsf{vk}, u, \pi_1) = \mathcal{V}(\mathsf{vk}, u, \pi_2) = 1 \wedge \pi_1 \neq \pi_2] \approx 0.$$

7.2.3 Composing Straight-Line Reductions

In this section, we show that straight-line extractability is closed under sequential and parallel composition. At a high-level, this holds because the aggregate extractor, which takes as input the aggregate transcript and query-response pairs, can appropriately partition the transcript and queries into two disjoint sets and pass each to the corresponding extractor.

Lemma 7.3 (Closure of Straight-Line Extractability). Straight-line extractability is closed under sequential and parallel composition.

Proof. Straight-line extractability of two straight-line protocols composed in parallel follows from the proof the of parallel composition theorem (Theorem 2.2). Essentially, the respective extractors are run independently in parallel.

We focus on proving straight-line extractability of two sequentially composed straight-line protocols. Let $\Pi_i = (\mathcal{G}, \mathcal{K}, \mathcal{P}_i, \mathcal{V}_i)$ for $i \in \{1, 2\}$. Let $\Pi_2 \circ \Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$.

We must construct a deterministic polynomial-time extractor $\mathcal{E}$ that satisfies the following property. For any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$ suppose $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$, $(\mathsf{s}, u_1, \mathsf{st}) \leftarrow \mathcal{A}$, $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$ and

$$(u_3, w_3) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st}).$$

Suppose additionally that the interaction between $\mathcal{P}^*$ and $\mathcal{V}$ produces corresponding transcript tr and prover query-response pairs qr to ρ. Then, we must show that

$$\Pr[(\mathsf{pp}, u_1, \mathcal{E}(\mathsf{pp}, \mathsf{s}, u_1, (\mathsf{tr}, \mathsf{qr}, w_3)))] = \epsilon - \mathsf{negl}(\lambda).$$

Indeed, let $\mathcal{E}_1$ and $\mathcal{E}_2$ be the extractors guaranteed by the straight-line extractability property of Π_1 and Π_2 respectively. We construct $\mathcal{E}$ as follows.

$\underline{\mathcal{E}(\mathsf{pp}, \mathsf{s}, u_1, (\mathsf{tr}, \mathsf{qr}, w_3)) \to w_1}$

1. Let tr_i and qr_i denote the portion of the transcript and query-response pairs associated with verifier $\mathcal{V}_i$ for $i \in \{1, 2\}$.
2. By the public reducibility property of Π_1 compute $u_2 \leftarrow \varphi(\mathsf{pp}, u_1, \mathsf{tr})$.
3. Compute $w_2 \leftarrow \mathcal{E}_2(\mathsf{pp}, \mathsf{s}, u_2, (\mathsf{tr}_2, \mathsf{qr}_2, w_3))$.
4. Output $w_1 \leftarrow \mathcal{E}_1(\mathsf{pp}, \mathsf{s}, u_1, (\mathsf{tr}_1, \mathsf{qr}_1, w_2))$.

Now, consider and arbitrary expected polynomial-time adversary $\mathcal{P}^*$ that succeeds with probability ϵ and suppose $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$, $(\mathsf{s}, u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$ and $(u_2, w_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})$ with corresponding transcript tr and the prover's query-response pairs qr to ρ. We must show that $\mathcal{E}$ succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$.

Indeed, because $\mathcal{P}^*$ succeeds with probability ϵ, we must have that $\mathcal{P}^*_1$ and $\mathcal{P}^*_2$, which represent the portion of $\mathcal{P}^*$ that interact with $\mathcal{V}_1$ and $\mathcal{V}_2$ respectively, both succeed with probability at least ϵ. Therefore, by the straight-line extractability of Π_1 and Π_2 we have that $\mathcal{E}_1$ and $\mathcal{E}_2$ each succeed with probability at least $\epsilon - \mathsf{negl}(\lambda)$. Therefore, we have that $\mathcal{E}$ succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$. □

7.2.4 A Straight-Line Opening Transformation

We now formally construct the straight-line opening transformation.

Construction 7.4 (Straight-Line Opening Transformation). Let ρ denote a random oracle that produces $n = O(\lambda)$ bit strings. Let $(\mathsf{com}, \Pi_{\mathsf{poly}} = (\mathcal{G}_{\mathsf{poly}}, \mathcal{K}_{\mathsf{poly}}, \mathcal{P}_{\mathsf{poly}}, \mathcal{V}_{\mathsf{poly}}))$ be a non-interactive, deterministic polynomial commitment scheme (Definition 4.10) with computationally injective proofs (Definition 7.2) in the random oracle model. For parameters $b, T, d \in \mathbb{N}$ denoting the hardness parameter, iteration bound, and polynomial degree bound respectively, such that $n - b = O(\lambda)$, $d = \mathsf{poly}(\lambda)$, and $T = O(\lambda \cdot 2^b)$, we construct a straight-line non-interactive proof of knowledge $\Pi_{\mathsf{open}} = (\mathcal{G}_{\mathsf{open}}, \mathcal{K}_{\mathsf{open}}, \mathcal{P}_{\mathsf{open}}, \mathcal{V}_{\mathsf{open}})$ for the knowledge of opening relation $\mathsf{Open}_{\mathsf{com}}$ (Definition 4.11) in the random oracle model as follows.

$\underline{\mathcal{G}_{\mathsf{open}}(\lambda, d) \to \mathsf{pp}}$:

1. Output $\mathsf{pp} \leftarrow \mathcal{G}_{\mathsf{poly}}(\lambda, d)$

$\underline{\mathcal{K}_{\mathsf{open}}(\mathsf{pp}) \to \mathsf{pp}}$:

1. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}_{\mathsf{poly}}(\mathsf{pp})$

$\underline{\mathcal{P}_{\mathsf{open}}(\mathsf{pk}, \overline{w}, w)}$:

1. For up to T iterations, repeatedly sample $x \in \mathbb{F}$ and compute $h \leftarrow \rho(\overline{w}, x, w(x), \pi)$ where $\pi \leftarrow \mathcal{P}_{\mathsf{poly}}(\mathsf{pk}, (\overline{w}, x, w(x)), w)$ until $h \leq 2^b$. Abort if no such h is found.
2. Send (x, y, π) to $\mathcal{V}_{\mathsf{open}}$ where $y \leftarrow w(x)$.

$\underline{\mathcal{V}_{\mathsf{open}}(\mathsf{vk}, \overline{w})}$:

1. Receive (x, y, π) from $\mathcal{P}_{\mathsf{open}}$.
2. Check that $\rho(\overline{w}, x, y, \pi) \leq 2^b$.
3. Check that $\mathcal{V}_{\mathsf{poly}}(\mathsf{pk}, (\overline{w}, x, w(x)), \pi)$ accepts.

Lemma 7.4 (Straight-Line Opening Transformation). Construction 7.4 transforms a non-interactive, deterministic polynomial commitment scheme $\mathsf{com} = ((\mathrm{gen}, \mathrm{com}), \Pi_{\mathsf{poly}})$ with computationally injective proofs in the random oracle model, into a straight-line non-interactive proof of knowledge Π_{open} for $\mathsf{Open}_{\mathrm{com}}$ in the random oracle model with the same communication complexity.

We prove Lemma 7.4 by proving the following lemmas which consider completeness and knowledge soundness separately.

Lemma 7.5 (Completeness). Construction 7.4 is complete.

Proof. Completeness holds due to the completeness of Π_{poly} so long as $\mathcal{P}_{\mathsf{open}}$ is able to produce a valid $h \leq 2^b$. We show that this is the case with probability $1 - \mathsf{negl}(\lambda)$.

Indeed, let $\mathtt{fail}$ denote the event that the prover is unable to produce a valid h. For $T = \lambda \cdot 2^b$, the probability of the prover failing can be bounded as follows:

$$\Pr[\mathtt{fail}] = \left(1 - \frac{1}{2^{n-b}}\right)^T \approx \frac{1}{e^\lambda} \leq \frac{1}{2^\lambda}$$

where n is the number of bits produced by each random oracle query. Thus, we have that the prover manages to find a valid $h \leq 2^b$ with probability $1 - \mathsf{negl}(\lambda)$. □

Lemma 7.6 (Knowledge-Soundness). Construction 7.4 is straight-line extractable.

Proof. To prove straight-line extractability we must construct extractor $\mathcal{E}$ that satisfies the following condition. Consider arbitrary expected-polynomial-time adversaries $\mathcal{A}_{\mathsf{open}}$ and $\mathcal{P}^*_{\mathsf{open}}$. Suppose $\mathsf{pp} \leftarrow \mathcal{G}_{\mathsf{open}}(\lambda, d)$ and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp})$. Suppose that for $(\overline{w}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$ we have that

$$\Pr[(\mathsf{pp}, \langle \mathcal{P}^*_{\mathsf{open}}, \mathcal{V}_{\mathsf{open}} \rangle((\mathsf{pk}, \mathsf{vk}), \overline{w}, \mathsf{st})) \in \mathcal{R}_\top] = \epsilon \tag{7.1}$$

Suppose that the interaction $\langle \mathcal{P}^*_{\mathsf{open}}, \mathcal{V} \rangle$ produces transcript tr and query-response pairs (with unique responses) qr. To prove straight-line extractability, we must show that

$$\Pr[(\mathsf{pp}, \overline{w}, \mathcal{E}(\mathsf{pp}, \overline{w}, \mathsf{tr}, \mathsf{qr}))] = \epsilon - \mathsf{negl}(\lambda). \tag{7.2}$$

Indeed, Equation (7.1) implies that $\mathcal{P}^*_{\mathsf{open}}$ produces message $(\overline{w}, x, y, \pi)$ such that

$$\mathcal{V}_{\mathsf{open}}(\mathsf{pp}, (\overline{w}, x, y), \pi) = 1 \tag{7.3}$$

and

$$\rho(\overline{w}, x, y, \pi) \leq 2^b. \tag{7.4}$$

with probability ϵ.

The probability that a queried message satisfying Equation (7.3) satisfies Equation (7.4) is

$$\frac{1}{2^{n-b}}.$$

Therefore, the probability that the prover successfully produces a message that satisfies both Equations (7.3) and (7.4) in less than $d + 1$ queries is at most

$$\frac{d}{2^{n-b}} = O\left(\frac{1}{2^\lambda}\right) = \mathsf{negl}(\lambda)$$

Moreover, for two query response pairs,

$$(h_1, ((\overline{w}, x_1, y_1, \pi_1)))$$

and

$$(h_2, ((\overline{w}, x_2, y_2, \pi_2)))$$

such that $h_1 \neq h_2$ we must have $x_i \neq x_j$ with probability $1 - \mathsf{negl}()$. In particular, $h_1 \neq h_2$ implies with probability $1 - \mathsf{negl}(\lambda)$ that $x_1 \neq x_2$, $y_2 \neq y_2$, or $\pi_1 \neq \pi_2$. If $y_1 \neq y_2$ this means that $x_1 \neq x_2$ by the knowledge soundness of Π_{poly}. Alternatively, if $\pi_1 \neq \pi_2$ by the computationally injective proofs property of Π_{poly}, we have that $x_1 \neq x_2$. Therefore, $\mathcal{P}^*_{\mathsf{open}}$ makes at least $d+1$ queries that satisfy Equation (7.3), each with unique values for x, with probability $1 - \mathsf{negl}(\lambda)$.

Then, the extractor on input qr can isolate any such $d+1$ values in qr with probability $\epsilon - \mathsf{negl}(\lambda)$ by checking Equation (7.3) and ignoring duplicate values of x. The extractor can then interpolate over these $d+1$ pairs of (x, y) values to retrieve a degree d opening polynomial w. Then, by the knowledge-soundness of Π_{poly} and by the binding property of com we have that

$$(\mathsf{pp}, \overline{w}, w) \in \mathsf{Open}_{\mathsf{com}}.$$

Therefore, we have that Equation (7.2) holds. □

7.2.5 A Straight-Line Transformation

We are now ready to present our straight-line opening transformation. Suppose we are provided with a (non-interactive) reduction of knowledge in the random oracle model $\Pi : \mathcal{R}_{\mathsf{com}} \to \mathcal{S}$ where $\mathcal{R}_{\mathsf{com}}$ is the committed relation (Definition 4.12) for relation $\mathcal{R}$, and a straight-line opening proof Π_{open} for com. Note that the latter is formally a proof of knowledge for relation $\mathsf{Open}_{\mathsf{com}}$ (Definition 4.11). We produce a straight-line (non-interactive) reduction of knowledge $\Pi' : \mathcal{R}_{\mathsf{com}} \to \mathcal{S}$ in the random oracle model.

At a high level, given instance $(u, \overline{w})$, we have the prover demonstrate knowledge of $\overline{w}$ using Π_{open} before running Π normally. Roughly, by the straight-line extractability of Π_{open}, we have that a straight-line extractor can produce an opening witness w to $\overline{w}$, and by the knowledge soundness of Π and the binding property of com, we have that w additionally satisfies the instance $(u, \overline{w})$ in $\mathcal{R}_{\mathsf{com}}$. We refer to Construction 7.3 for additional exposition of the straight-line transformation.

Construction 7.5 (Straight-Line Transformation). Let ρ denote a random oracle. Let $\mathsf{com} = (\mathrm{gen}, \mathrm{com})$ be a commitment scheme such that there exists a straight-line, non-interactive proof of knowledge $\Pi_{\mathsf{open}} = (\mathcal{G}_{\mathsf{open}}, \mathcal{K}_{\mathsf{open}}, \mathcal{P}_{\mathsf{open}}, \mathcal{V}_{\mathsf{open}})$ for $\mathsf{Open}_{\mathrm{com}}$ in the random oracle model.

Consider a (non-interactive) reduction of knowledge $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_{\mathsf{com}} \overset{\sim}{\to} \mathcal{S}$ in the random oracle model. We construct a (non-interactive) straight-line reduction of knowledge $\Pi' = (\mathcal{G}', \mathcal{K}', \mathcal{P}', \mathcal{V}') : \mathcal{R}_{\mathsf{com}} \overset{\sim}{\to} \mathcal{S}$ in the random oracle model as follows.

$\underline{\mathcal{G}'(\lambda, n) \to \mathsf{pp}}$:

1. Compute and output $\mathsf{pp} \leftarrow (\mathsf{pp}', \mathsf{pp}_{\mathsf{open}}) \leftarrow (\mathcal{G}(\lambda, n), \mathcal{G}_{\mathsf{open}}(\lambda, n))$.

$\underline{\mathcal{K}'(\mathsf{pp}, \mathsf{s}) \to (\mathsf{pk}, \mathsf{vk})}$:

1. Compute $(\mathsf{pk}', \mathsf{vk}') \leftarrow \mathcal{K}(\mathsf{pp}', \mathsf{s})$
2. Compute $(\mathsf{pk}_{\mathsf{open}}, \mathsf{vk}_{\mathsf{open}}) \leftarrow \mathcal{K}_{\mathsf{open}}(\mathsf{pp}_{\mathsf{open}})$.
3. Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((\mathsf{pk}', \mathsf{pk}_{\mathsf{open}}), (\mathsf{vk}', \mathsf{vk}_{\mathsf{open}}))$.

$\underline{\mathcal{P}'(\mathsf{pk}, (u, \overline{w}), (w, r)) \rightarrow (u_2, w_2)}$:

1. Run $\mathcal{P}_{\mathsf{open}}(\mathsf{pk}_{\mathsf{open}}, \overline{w}, w)$.
2. Compute and output $(u_2, w_2) \leftarrow \mathcal{P}(\mathsf{pk}', (u, \overline{w}), (w, r))$.

$\underline{\mathcal{V}'(\mathsf{vk}, (u, \overline{w})) \rightarrow u_2}$:

1. Run $\mathcal{V}_{\mathsf{open}}(\mathsf{vk}_{\mathsf{open}}, \overline{w})$.
2. Compute and output $u_2 \leftarrow \mathcal{V}(\mathsf{vk}', (u, \overline{w}))$.

Theorem 7.2 (Straight-Line Transformation). Let $\mathsf{com} = (\mathrm{gen}, \mathrm{com})$ be a commitment scheme such that there exists a straight-line, non-interactive proof of knowledge Π_{open} for $\mathsf{Open}_{\mathrm{com}}$ in the random oracle model. Construction 7.5 transforms a (non-interactive) reduction of knowledge $\Pi : \mathcal{R}_{\mathsf{com}} \overset{\sim}{\rightarrow} \mathcal{S}$ in the random oracle model into a (non-interactive) straight-line reduction of knowledge $\Pi' : \mathcal{R}_{\mathsf{com}} \overset{\sim}{\rightarrow} \mathcal{S}$ in the random oracle model. The communication complexity of Π' is equal to the combined communication complexity of Π_{open} and Π.

Proof. Completeness follows from the completeness of Π_{open} and Π. Public reducibility follows from the public reducibility of Π.

Let $\Pi' = (\mathcal{G}', \mathcal{K}', \mathcal{P}', \mathcal{V}')$. We now prove straight-line extractability. We must construct an extractor $\mathcal{E}$ that satisfies the following property. Consider expected-polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$. Suppose that for $\mathsf{pp} \leftarrow \mathcal{G}'(\lambda, n)$, $((\mathsf{s}, u, \overline{w}), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}'(\mathsf{pp}, \mathsf{s})$ we have that

$$\Pr[(\mathsf{pp}, \langle \mathcal{P}^*, \mathcal{V}' \rangle((\mathsf{pk}, \mathsf{vk}), (u, \overline{w}), \mathsf{st})) \in \mathcal{S}] = \epsilon.$$

Suppose that the interaction $\langle \mathcal{P}^*_{\mathsf{open}}, \mathcal{V}_{\mathsf{open}} \rangle$, produces prover output witness w_2, transcript tr, and prover query-response pairs qr to ρ. Then, we must have that

$$\Pr[(\mathsf{pp}, (u, \overline{w}), \mathcal{E}(\mathsf{pp}, (u, \overline{w}), (\mathsf{tr}, \mathsf{qr}, w_2))) \in \mathcal{R}_{\mathsf{com}}] = \epsilon - \mathsf{negl}(\lambda)$$

At a high-level, we first use $\mathcal{A}$ and $\mathcal{P}^*$ to construct corresponding adversaries $\mathcal{A}_{\mathsf{open}}$ and $\mathcal{P}^*_{\mathsf{open}}$ for the proof of commitment opening portion of the overall proof of knowledge. We then invoke the straight-line extractability of Π_{open} to obtain a corresponding straight-line extractor $\mathcal{E}_{\mathsf{open}}$ that extracts the witness underlying the commitment. Using $\mathcal{E}_{\mathsf{open}}$, we construct the desired extractor $\mathcal{E}$, which simply outputs the output of $\mathcal{E}_{\mathsf{open}}$. We then invoke the binding property of the commitment scheme, and the knowledge-soundness of Π to argue that the witness produced by $\mathcal{E}$ further satisfies the overall relation $\mathcal{R}_{\mathsf{com}}$.

Indeed, using $\mathcal{A}$ and $\mathcal{P}^*$, we first construct the corresponding expected-polynomial-time adversaries $\mathcal{A}_{\mathsf{open}}$ and $\mathcal{P}^*_{\mathsf{open}}$ for the proof of knowledge Π_{open} for $\mathsf{Open}_{\mathrm{com}}$.

$\underline{\mathcal{A}_{\mathsf{open}}(\mathsf{pp}_{\mathsf{open}}) \rightarrow (\overline{w}, \mathsf{st})}$:

1. Compute $\mathsf{pp}' \leftarrow \mathcal{G}(\lambda, n)$ and let $\mathsf{pp} \leftarrow (\mathsf{pp}', \mathsf{pp}_{\mathsf{open}})$.
2. Compute $((u, \overline{w}), \mathsf{st}') \leftarrow \mathcal{A}(\mathsf{pp})$ and let $\mathsf{st} \leftarrow (u, \mathsf{st}', \mathsf{pp}')$.
3. Output $(\overline{w}, \mathsf{st})$.

$\underline{\mathcal{P}^*_{\mathsf{open}}(\mathsf{pp}_{\mathsf{open}}, \overline{w}, \mathsf{st}) \to \bot}$:

1. Parse $(u, \mathsf{st}', \mathsf{pp}') \leftarrow \mathsf{st}$ and let $\mathsf{pp} \leftarrow (\mathsf{pp}', \mathsf{pp}_{\mathsf{open}})$
2. Run $\mathcal{P}^*(\mathsf{pp}, (u, \overline{w}), \mathsf{st}')$ until it produces a single message m intended for $\mathcal{V}_{\mathsf{open}}$ internal to $\mathcal{V}'$.
3. Send m to the verifier.

By observation, we have that $\mathcal{A}_{\mathsf{open}}$ and $\mathcal{P}^*_{\mathsf{open}}$ succeed with the same probability as $\mathcal{A}$ and $\mathcal{P}^*$. Then, for $\mathsf{pp}_{\mathsf{open}} \leftarrow \mathcal{G}(\lambda, n)$, $(\overline{w}, \mathsf{st}) \leftarrow \mathcal{A}_{\mathsf{open}}(\mathsf{pp}_{\mathsf{open}})$, $(\mathsf{pk}_{\mathsf{open}}, \mathsf{vk}_{\mathsf{open}}) \leftarrow \mathcal{K}(\mathsf{pp}_{\mathsf{open}})$ we have that

$$\Pr[(\mathsf{pp}, \langle \mathcal{P}^*_{\mathsf{open}}, \mathcal{V}_{\mathsf{open}} \rangle((\mathsf{pk}_{\mathsf{open}}, \mathsf{vk}_{\mathsf{open}}), \overline{w}, \mathsf{st})) \in \mathcal{R}_\top] = \epsilon.$$

Then, by the straight-line extractibility of the opening proof, we have that there exists a corresponding straight-line deterministic extractor $\mathcal{E}_{\mathsf{open}}$ that succeeds in producing an opening for $\overline{w}$ with nearly the same probability. Formally, given that the interaction $\langle \mathcal{P}^*_{\mathsf{open}}, \mathcal{V}_{\mathsf{open}} \rangle$ produces transcript tr and prover query-response pairs qr to ρ, we have that

$$\Pr[(\mathsf{pp}_{\mathsf{open}}, \overline{w}, \mathcal{E}_{\mathsf{open}}(\mathsf{pp}_{\mathsf{open}}, \overline{w}, (\mathsf{tr}, \mathsf{qr}))) \in \mathsf{Open}_{\mathsf{com}}] = \epsilon - \mathsf{negl}(\lambda). \tag{7.5}$$

Using $\mathcal{E}_{\mathsf{open}}$, we can construct the desired straight-line deterministic extractor $\mathcal{E}$ as follows.

$\underline{\mathcal{E}(\mathsf{pp}, (u, \overline{w}), (\mathsf{tr}, \mathsf{qr}, w_2)) \to w}$:

1. Parse $(\mathsf{pp}', \mathsf{pp}_{\mathsf{open}}) \leftarrow \mathsf{pp}$.
2. Let tr' be the single message m in tr intended for $\mathcal{V}_{\mathsf{open}}$ internal to $\mathcal{V}'$.
3. Likewise, let qr' be the the portion of query-response pairs in qr made by $\mathcal{P}^*$ before sending the single message m intended for $\mathcal{V}_{\mathsf{open}}$.
4. Compute and output $(w, r) \leftarrow \mathcal{E}_{\mathsf{open}}(\mathsf{pp}_{\mathsf{open}}, \overline{w}, (\mathsf{tr}', \mathsf{qr}'))$.

We now analyze the success probability of $\mathcal{E}$. By observation, we have that $\mathcal{E}$ provides the same distribution of inputs to $\mathcal{E}_{\mathsf{open}}$ as was provided in Equation (7.5). Therefore, we must have that the witness (w, r) produced by $\mathcal{E}$ is an opening to $\overline{w}$ with probability $\epsilon - \mathsf{negl}(\lambda)$.

We must still argue that the witness (w, r) produced by $\mathcal{E}$ satisfies the overall relation. Formally, we need to show that $(\mathsf{pp}, \mathsf{s}, (u, \overline{w}), (w, r)) \in \mathcal{R}_{\mathsf{com}}$. To argue this, we first observe that using $\mathcal{A}$ and $\mathcal{P}^*$ we can construct expected-polynomial-time adversaries $\mathcal{A}_\Pi$ and $\mathcal{P}^*_\Pi$ for $\Pi : \mathcal{R}_{\mathsf{com}} \to \mathcal{S}$ that succeed with probability ϵ much in the same manner as we did

to construct $\mathcal{A}_{\mathsf{open}}$ and $\mathcal{P}^*_{\mathsf{open}}$. Then, by the knowledge-soundness of Π there exists a corresponding expected-polynomial-time extractor $\mathcal{E}_\Pi$ (not necessarily straight-line) that produces a witness (w', r') such that

$$(\mathsf{pp}, \mathsf{s}, (u, \overline{w}), (w', r')) \in \mathcal{R}_{\mathsf{com}}. \tag{7.6}$$

Then, we must necessarily have that (w, r) produced by $\mathcal{E}$ is equal to (w', r') produced by $\mathcal{E}_\Pi$ because otherwise an adversary can use both $\mathcal{E}$ and $\mathcal{E}_\Pi$ to break the binding property of com.

Therefore, by Equation (7.6), for $\mathsf{pp} \leftarrow \mathcal{G}'(\lambda, n)$, $(\mathsf{s}, (u, \overline{w}), \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$, $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}'(\mathsf{pp}, \mathsf{s})$ and given prover output w_2, interaction transcript tr, and prover query-response pairs qr in the interaction $\langle \mathcal{P}^*, \mathcal{V}' \rangle(\mathsf{pp}, (u, \overline{w}), \mathsf{st})$ we have that

$$\Pr[(\mathsf{pp}, (u, \overline{w}), \mathcal{E}(\mathsf{pp}, (u, \overline{w}), (\mathsf{tr}, \mathsf{qr}, w_2))) \in \mathcal{R}_{\mathsf{com}}] = \epsilon - \mathsf{negl}(\lambda).$$

□

7.3 A Zero-Knowledge Transformation

In this section, we initiate the study of *blinding* folding schemes, that is, folding schemes that can be used to randomize witnesses for complex relations. The central insight of this section is that much in the same way we can blind *linear* statements using a *random linear combination* we can blind *non-linear* statements using folding schemes. We demonstrate that a blinding folding scheme for a relation enables us to generically transform any reduction of knowledge for that relation into a zero-knowledge reduction of knowledge for that relation, which first blinds the witness and then runs the standard reduction of knowledge.

The main benefit is that we can perform the blinding step *external* to the reduction (i.e., before or after), as opposed to *internal* the reduction (i.e., during interaction). This affords practical savings, but also significantly eases the implementation effort: Instead of carefully implementing a zero-knowledge variant of every step of the reduction, which may include complex subroutines such as the sumcheck protocols, practitioners can simply prepend a blinding folding scheme to a simpler non-zero-knowledge reduction.

Since the inception of zero-knowledge proofs of knowledge, there has been a sustained interest in transforming traditional proofs of knowledge into zero-knowledge variants. Such transformations abstract away details specific to achieving zero-knowledge, and often provides a starting point for subsequent optimizations.

The first feasibility result is due to Ben-Or et al. [22] who observe that for any proof of knowledge, the verifier itself can be treated as a polynomial-time predicate enabling a corresponding zero-knowledge proof of knowledge where statements about the original verifier accepting are proven using a zero-knowledge proof of knowledge for NP [83].

The definitive zero-knowledge transformation, due to Cramer and Damgard [60], considers a restricted — but natural — class of *algebraic* proofs of knowledge. In particular, the transformation assumes that the the prover only sends finite field elements to the verifier, which in turn only performs simple algebraic checks on these elements. Then, to

achieve zero-knowledge, the prover sends homomorphic hiding commitments to these field elements, and then demonstrates that the original algebraic checks hold for the underlying values using standard zero-knowledge proofs for basic algebraic relations.

Unfortunately, the Cramer-Damgard transformation, which by far is the most efficient to date, still remains too expensive to apply generically in the setting of high-performance proofs of knowledge. In practice, modern zero-knowledge proofs must often start with the general template outlined by Cramer and Damgard, before carefully hand-tuning each round of interaction [136]. This complicates both the soundness reasoning on paper and implementation effort in practice.

In this section, we develop a zero-knowledge transformation that retains the generality of the Cramer-Damgard transformation, while remaining acceptable for high performance applications. Our central approach is that instead of blinding each message of the interaction, we can directly blind the original statement and witness. This allows our transformation to be completely agnostic to the underlying proof of knowledge.

Such an approach has been previously explored for *linear* statements: For instance, Chiesa et al. [53] develop a zero-knowledge sumcheck protocol (i.e., a proof of knowledge for proving the sum of evaluations of a witness polynomial) by blinding the witness polynomial with a randomized polynomial before running the standard sumcheck interaction. Boneh et al. [35], similarly develop a zero-knowledge homomorphism preimage evaluation proof (i.e., a proof demonstrating knowledge of a preimage to the evaluation of a homomorphism) by initially blinding the preimage with a randomized vector. Both works rely on linearity to ensure that blinding the original statement-witness pair with a randomized statement-witness pair does not break satisfiablity.

In the general setting, however, it is not immediately clear how to extend this approach for *non-linear* relations such as circuit-satisfiability. Our central insight is that in the non-linear setting, a folding scheme (with mild qualifications) can play the role of the blinding operation to randomize the original statement-witness pair in a way that perserves satisfiability. Recall that folding schemes are interactive proofs that reduce the task of checking two statements (possibly in an NP-complete relation) to checking a single statement of the same size (Definition 4.1). Of course, our transformation, then, only applies to relations equipped with such a folding scheme. While, folding schemes are a nascient primitive, preliminary evidence suggests that they are often easier to achieve than full proofs of knowledge [35, 45, 101]. Indeed, we demonstrate that all linear relations as well as R1CS (a generalization of circuit-satisfiablity [78]) have efficient folding schemes that satisfy our desiderata.

In more detail, our transformation requires a folding scheme that is *blinding*. A folding scheme is blinding if (1) a randomized statement-witness pair can be sampled efficiently, and (2) folding this randomized statement-witness pair into the original statement-witness pair produces a new statement-witness pair that is jointly simulatable alongside the verifier's view.

Given such a folding scheme, our transformation more broadly converts any reduction of knowledge into a zero-knowledge reduction of knowledge. Our transformation achieves *strong zero-knowledge*, a stricter notion of zero-knowledge that is only meaningful in the context of reductions of knowledge: Both the interaction *and* the prover's (reduced) output

witness reveal no information about the original witness.

Theorem 7.3 (Zero-Knowledge Transformation). There exists a transformation from a blinding folding scheme for relation $\mathcal{R}_1$ and a reduction of knowledge from relation $\mathcal{R}_1$ to relation $\mathcal{R}_2$ to a strong zero-knowledge reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$. The time, space and commmunication complexity is the combined time, space, and communication complexity of the underlying folding scheme and reduction of knowledge.

Unlike standard zero-knowledge, which is only preserved when all steps of a proof of knowledge are zero-knowledge (Lemma 7.8), strong zero-knowledge is a remarkably robust property that is preserved even when composed with a non-zero-knowledge reduction.

Lemma 7.7 (Preservation of Strong Zero-Knowledge, Informal). Consider a reduction of knowledge $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3$ and a strong zero-knowledge reduction of knowledge $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$. Then $\Pi_2 \circ \Pi_1$ is a strong zero-knowledge reduction of knowledge, where $\Pi_2 \circ \Pi_1$ denotes the protocol that first runs Π_1, and then runs Π_2 on the statement and witness output by Π_2.

The remainder of this section formally treats all of the introduced concepts. In Section 7.3.1, we concretely demonstrate each component of our transformation: We first demonstrate that the folding scheme for committed relaxed R1CS (Construction 4.1) is a blinding folding scheme, and then demonstrate that this implies a strong zero-knowledge proof for this relation. In Section 7.3.2 we define (strong) zero-knowledge, and in Section 7.3.3 and prove that zero-knowledge is closed under composition and that strong zero-knowledge is preserved even when composed with non-zero-knowledge reductions. In Section 7.3.4, we define blinding folding schemes and formally construct our zero-knowledge transformation. In Section 7.3.5, we demonstrate blinding folding schemes for both linear and non-linear relations and discuss several applications.

7.3.1 Overview

In this section, we demonstrate that the folding scheme for committed relaxed R1CS is blinding. As a result, we show that we can make any proof system for relaxed R1CS (and thus NP) zero-knowledge. Our high-level strategy is as follows: First, we recall relaxed R1CS. Next, we show how to randomly sample satisfying relaxed R1CS instance-witness pairs. We then show how the prover and verifier can interactively fold this randomly sampled pair into an existing instance-witness pair to blind it. We then argue that the prover and verifier are free to use any proof of knowledge to demonstrate satisfiability of this blinded instance witness pair and still preserve zero-knowledge.

Indeed, recall that R1CS [78] is a popular NP-complete constraint system. Recall from Section 4.2 that relaxed R1CS a folding-friendly variant of R1CS. A relaxed R1CS instance consists of constraint matrices $A, B, C \in \mathbb{F}^{m \times m}$, public inputs and outputs vector x, scalar u, and error vector E. A witness W (representing variable assignments) is satisfying if for $Z = (W, x, u)$ we have that $AZ \circ BZ = u \cdot CZ + E$. Recall from Section 4.2 that to enable a succinct verifier, we relegate E to the witness and instead have the statement refer to Pedersen commitments $\overline{E}$ and $\overline{W}$ to E and W respectively.

In particular, consider an additively homomorphic commitment scheme com (e.g., the Pedersen commitment scheme [119]) and appropriately sampled public parameters pp. Consider constraint matrices $(A, B, C) \in \mathbb{F}^{m \times m}$, public input and output vector x, scalar $u \in \mathbb{F}$, and commitments $\overline{E}$ and $\overline{W}$. Suppose now that the prover would like to demonstrate in zero-knowledge to the verifier that it knows vectors E and W (along with commitment randomness r_E and r_W) such that for $Z = (W, \mathsf{x}, u)$

$$AZ \circ BZ = u \cdot CZ + E$$

and that $\overline{E} = \mathsf{com}(\mathsf{pp}, E, r_E)$ and $\overline{W} = \mathsf{com}(\mathsf{pp}, W, r_W)$. While the prover can directly send E and W (and the corresponding commitment randomness) for the verifier to check directly, this would sacrifice zero-knowledge.

Instead, the prover begins by sampling a *randomized* relaxed R1CS instance-witness pair as follows:

1. Sample random $(W_{\mathsf{blind}}, \mathsf{x}_{\mathsf{blind}}, u_{\mathsf{blind}}) \xleftarrow{\$} \mathbb{F}^m$ and $(r_{E_{\mathsf{blind}}}, r_{W_{\mathsf{blind}}}) \xleftarrow{\$} \mathbb{F}^2$
2. Letting $Z_{\mathsf{blind}} = (W_{\mathsf{blind}}, \mathsf{x}_{\mathsf{blind}}, u_{\mathsf{blind}})$, solve for the appropriate error term $E_{\mathsf{blind}} \leftarrow AZ_{\mathsf{blind}} \circ BZ_{\mathsf{blind}} - u_{\mathsf{blind}} \cdot CZ_{\mathsf{blind}}$
3. Compute the corresponding commitments

$$\begin{aligned} \overline{E}_{\mathsf{blind}} &\leftarrow \mathsf{com}(\mathsf{pp}, E_{\mathsf{blind}}, r_{E_{\mathsf{blind}}}) \\ \overline{W}_{\mathsf{blind}} &\leftarrow \mathsf{com}(\mathsf{pp}, W_{\mathsf{blind}}, r_{W_{\mathsf{blind}}}). \end{aligned}$$

The prover then sends the instance $(\mathsf{x}_{\mathsf{blind}}, u_{\mathsf{blind}}, \overline{E}_{\mathsf{blind}}, \overline{W}_{\mathsf{blind}})$ to the verifier.

The prover's strategy now is to use the randomized witness $(W_{\mathsf{blind}}, E_{\mathsf{blind}}, r_{W_{\mathsf{blind}}}, r_{E_{\mathsf{blind}}})$ to blind the original witness (W, E, r_W, r_E). Unfortunately, because the statement enforces non-linear constraints on the witness, the prover cannot simply take a random linear combination of these terms. Instead, the prover and verifier can run an interactive folding proof to reduce the task of checking the original witness (and the blinding witness) into the task of checking a new randomized witness.

In particular, The prover begins by sending a commitment $\overline{T} \leftarrow \mathsf{com}(\mathsf{pp}, T, r_T)$ to all the cross terms for

$$T \leftarrow AZ \circ BZ_{\mathsf{blind}} + AZ_{\mathsf{blind}} \circ BZ - u \cdot CZ_{\mathsf{blind}} - u_{\mathsf{blind}} \cdot CZ$$

for randomly sampled r_T. In response, the verifier sends a challenge $r \xleftarrow{\$} \mathbb{F}$.

Together, the prover and verifier compute a new blinded instance

$$\begin{aligned} \mathsf{x}' &\leftarrow \mathsf{x} + r \cdot \mathsf{x}_{\mathsf{blind}}, \\ u' &\leftarrow u + r \cdot u_{\mathsf{blind}}, \\ \overline{E}' &\leftarrow \overline{E} + r \cdot \overline{T} + r^2 \cdot \overline{E}_{\mathsf{blind}}, \\ \overline{W}' &\leftarrow \overline{W} + r \cdot \overline{W}_{\mathsf{blind}}. \end{aligned}$$

Privately, the prover computes a new blinded witness $E' \leftarrow E + r \cdot T + r^2 \cdot E_{\mathsf{blind}}$, $r'_E \leftarrow r_E + r \cdot r_T + r^2 \cdot r_{E_{\mathsf{blind}}}$, $W' \leftarrow W + r \cdot W_{\mathsf{blind}}$, and $r'_W \leftarrow r_W + r \cdot r_{W_{\mathsf{blind}}}$.

By Theorem 4.1, we have that completeness and soundness holds for this scheme. Critically, we must now argue that the output blinded witness reveals no information about the original witness. Intuitively, in the folding scheme itself, the prover only sends a random commitment, thereby ensuring no information about the witness is leaked in the blinding process. Then, because the blinding instance-witness pair completely rerandomizes the original instance-witness pair, the blinded witness reveals no information about the original witness. Formally, we can construct a simulator that can faithfully simulate the joint distribution of the folding proof interaction transcript and the output instance-witness pair even without access to the original witness (thereby demonstrating that these values are independent of the original witness). Roughly, the simulator begins by sampling the output instance-witness pair and backsolving for the remainder of the transcript accordingly.

Now, the prover is free to use any proof of knowledge to demonstrate that it knows a valid witness for the blinded instance, which in turn implies that it knows a witness for the original instance. The simplest such proof involves the prover directly sending the blinded witness to the verifier, which can directly check it against the blinded instance. Alternatively, we can use the proof of knowledge for relaxed R1CS in Section 6.1.

7.3.2 Defining Zero-Knowledge

We now define (strong) zero-knowledge in a way that is composable. Recall that the standard definition of zero-knowledge dictates that for any malicious verifier $\mathcal{V}^*$ there exists a simulator $\mathcal{S}$ that can simulate an an interaction between $\mathcal{V}^*$ and an honest prover, for any input instance [42, 100, 124, 136]. As the simulator is not provided a witness, this captures the property that the interaction transcript is indeed independent of the witness.[1] We recall such a definition below.

Definition 7.3 (Zero-Knowledge, Traditional). A reduction $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ from $\mathcal{R}_1$ to $\mathcal{R}_2$ satisfies *zero-knowledge* if for any PPT adversary $\mathcal{V}^*$ there exists an EPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$ we have that

$$\left\{ \ \mathsf{tr} \ \middle| \ \mathsf{tr} \leftarrow \langle \mathcal{P}, \mathcal{V}^*(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \ \right\} \cong \left\{ \ \mathsf{tr} \ \middle| \ \mathsf{tr} \leftarrow \mathcal{S}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \ \right\}.$$

where tr denotes the interaction transcript.

This definition, however, becomes problematic in the context of composition. For a composed reduction $\Pi_2 \circ \Pi_1$, we could indeed have simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ that produce

[1] More subtly, the existence of a simulator that can produce an accepting transcript means that a proof of knowledge implies that either **(1)** the prover knows the witness or **(2)** that it could predict the verifier's randomness or alternatively knows a trapdoor in the non-interactive setting (as is required by the simulator). Because a prover cannot predict the verifier's randomness in the real world, a proof of knowledge indeed implies knowledge of a valid witness.

indistinguishable transcripts tr_1 and tr_2 for the first and second half of the protocol respectively. However, it is unclear if the joint distribution of $(\mathsf{tr}_1, \mathsf{tr}_2)$ is identical to that of a real interaction. To address this, Jones [90, Conjecture 4.1] conjectures that composability holds under a stronger definition in which the simulator $\mathcal{S}_2$ can simulate a transcript tr_2 to a distinguisher that is already provided tr_1. While this conjecture seems plausible, such a definition would require jointly reasoning about reductions Π_1 and Π_2 to assert that $\Pi_2 \circ \Pi_1$ is zero-knowledge.

To achieve modularity, we must formulate a definition of zero-knowledge that allows us to generically derive that $\Pi_2 \circ \Pi_1$ is zero-knowledge given that Π_1 and Π_2 are zero-knowledge. Returning to the traditional definition of zero-knowledge, let $\mathcal{V}_1^*$ and $\mathcal{V}_2^*$ denote the portions of $\mathcal{V}^*$ associated with Π_1 and Π_2 respectively. The central issue with the prior definition is that V_1^* may signal some additional secret state st_2 to $\mathcal{V}_2^*$ that causes it to shift its distribution of queries in a way that is detectable to the distinguisher. If $\mathcal{S}_2$ was provided this intermediate state st_2, then, by definition, it could produce a transcript tr_2 that is appropriately conditioned on tr_1. Unfortunately, the overall simulator $\mathcal{S}$ may not be able to simulate such an intermediate state st_2 that is consistent with tr_1 simulated by $\mathcal{S}_1$.

This discrepancy, however, illuminates the necessary modification to the traditional definition: Instead, of requiring the simulator simulates a transcript, we can instead require it to simulate the verifier's output state, as was written in the *original* definition of zero-knowledge by Goldwasser, Micali, and Rackoff [83]. In particular, we require that for any verifier $\mathcal{V}_1^*$, the simulator $\mathcal{S}_1$ simulate the verifier's output state st_2 given the verifier's input state st_1. Conceptually, as opposed to simulating the interaction itself, the simulator is required to simulate anything the verifier can say after the interaction.

Then, for any verifier $\mathcal{V}^*$ for protocol $\Pi_2 \circ \Pi_1$, a simulator $\mathcal{S}$, provided the initial state st_1, can first run $\mathcal{S}_1$ to simulate the intermediate state st_2 produced by $\mathcal{V}_1^*$. Then, $\mathcal{S}$ can run S_2 on input st_2 to simulate the state st_3 produced by $\mathcal{V}_3^*$, thus proving zero-knowledge for $\Pi_2 \circ \Pi_1$. We are now ready to formally define (strong) zero-knowledge.

Definition 7.4 (Zero-Knowledge). A reduction of knowledge $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ from $\mathcal{R}_1$ to $\mathcal{R}_2$ satisfies *zero-knowledge* if for any PPT adversary $\mathcal{V}^*$ there exists an EPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$ we have that

$$\left\{ \ \mathsf{st}_2 \ \middle| \ (\mathsf{st}_2, w_2) \leftarrow \langle \mathcal{P}, \mathcal{V}^*(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \ \right\} \cong \left\{ \ \mathsf{st}_2 \ \middle| \ \mathsf{st}_2 \leftarrow \mathcal{S}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \ \right\}.$$

Definition 7.5 (Honest-Verifier Zero-Knowledge). A reduction of knowledge satisfies honest-verifier zero-knowledge (HVZK) if it satisifes zero-knowledge under an honest (but curious) verifier.

Definition 7.6 (Strong Zero-Knowledge). A reduction of knowledge $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ from $\mathcal{R}_1$ to $\mathcal{R}_2$ satisfies strong zero-knowledge if for any PPT adversary $\mathcal{V}^*$ there exists an EPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$, given $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$ we have that

$$\left\{ \ (\mathsf{st}_2, w_2) \ \middle| \ (\mathsf{st}_2, w_2) \leftarrow \langle \mathcal{P}, \mathcal{V}^*(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \ \right\} \cong$$
$$\left\{ \ (\mathsf{st}_2, w_2) \ \middle| \ (\mathsf{st}_2, w_2) \leftarrow \mathcal{S}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \ \right\}.$$

7.3.3 Composing Zero-Knowledge Reductions

We now demonstrate that zero-knowledge is closed under sequential and parallel composition. We additionally show that prepending a strong zero-knowledge reduction with *any* reduction still results in a strong zero-knowledge reduction.

Lemma 7.8 (Closure under Sequential Composition). (Honest-verifier) zero-knowledge is closed under sequential composition.

Proof. Consider zero-knowledge reductions of knowledge $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_1, \mathcal{V}_1)$ and $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_2, \mathcal{V}_2)$. Consider PPT adversary $\mathcal{V}^*$ for $\Pi_2 \circ \Pi_1$. We must construct a corresponding EPT simulator $\mathcal{S}$ for any PPT adversary $\mathcal{A}$.

Indeed, by construction, we have that $\mathcal{P}$ first runs $\mathcal{P}_1$ to produce an intermediate statement and witness (u_2, w_2) and then runs $\mathcal{P}_2$ with (u_2, w_2). As such, we have that $\mathcal{V}^*$ first runs some $\mathcal{V}_1^*$ and then runs some $\mathcal{V}_2^*$ such that $\mathcal{V}_1^*$ interacts with $\mathcal{P}_1$ and then passes some state to $\mathcal{V}_2^*$ which interacts with $\mathcal{P}_2$ before the two parties collectively produce the output (st_3, w_3).

We construct PPT $\mathcal{A}_1$ and $\mathcal{V}_1^{**}$ as follows

$\underline{\mathcal{A}_1(\mathsf{pp}) \to (\mathsf{s}, u_1, w_1, \mathsf{st}_1)}$:

1. Compute and output $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$.

$\underline{\mathcal{V}_1^{**}(\mathsf{vk}, u_1, \mathsf{st}_1) \to \mathsf{st}_2}$:

1. Run $\mathcal{V}_1^*(\mathsf{vk}, u_1, \mathsf{st}_1)$, which, at the end of interaction produces st_2'. Record the corresponding transcript as tr_1
2. Output $(\mathsf{tr}_1, \mathsf{st}_2')$.

Then, by the zero-knowledge property of Π_1, we have that there exists EPT simulator $\mathcal{S}_1$ such that for $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$

$$\begin{aligned} &\left\{ \ \mathsf{st}_2 \ \middle| \ (\mathsf{st}_2, w_2) \leftarrow \langle \mathcal{P}_1, \mathcal{V}_1^{**}(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \ \right\} \cong \\ &\left\{ \ \mathsf{st}_2 \ \middle| \ \mathsf{st}_2 \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \ \right\}. \end{aligned} \tag{7.7}$$

Next, we construct PPT adversary $\mathcal{A}_2$ and $\mathcal{V}_2^{**}$ as follows.

$\underline{\mathcal{A}_2(\mathsf{pp}) \to (\mathsf{s}, u_2, w_2, \mathsf{st}_2)}$:

1. Compute $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$.
2. Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$
3. Compute $(\mathsf{st}_2, w_2) \leftarrow \langle \mathcal{P}_1, \mathcal{V}_1^*(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1)$. Let u_2 be the statement output by $\mathcal{P}_1$.
4. Output $(\mathsf{s}, u_2, w_2, \mathsf{st}_2)$.

$\underline{\mathcal{V}_2^{**}(\mathsf{vk}, u_2, \mathsf{st}_2) \to \mathsf{st}_3}$:

1. Run $\mathcal{V}_2^*(\mathsf{vk}, u_2, \mathsf{st}_2)$, which, at the end of interaction produces st_3.

2. Output st_3.

By the zero-knowledge property of Π_2, we have that there exists EPT simulator $\mathcal{S}_2$ such that for $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, u_2, w_2, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$

$$\begin{aligned} &\left\{ \ \mathsf{st}_3 \ \middle| \ (\mathsf{st}_3, w_3) \leftarrow \langle \mathcal{P}_2, \mathcal{V}_2^{**}(\mathsf{st}_2) \rangle ((\mathsf{pk}, \mathsf{vk}), u_2, w_2) \ \right\} \cong \\ &\left\{ \ \mathsf{st}_3 \ \middle| \ \mathsf{st}_3 \leftarrow \mathcal{S}_2(\mathsf{pp}, \mathsf{s}, u_2, \mathsf{st}_2) \ \right\}. \end{aligned} \tag{7.8}$$

We construct a simulator $\mathcal{S}$ for $\Pi_2 \circ \Pi_1$ as follows. Let φ_1 be the deterministic function guaranteed by the public reducibility of Π_1.

$\underline{\mathcal{S}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \to \mathsf{st}_3}$:

1. Compute $\mathsf{st}_2 \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1)$

2. Parse st_2 as $(\mathsf{tr}_1, \mathsf{st}_2')$

3. Compute $u_2 \leftarrow \varphi_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{tr}_1)$

4. Compute and output $\mathsf{st}_3 \leftarrow \mathcal{S}_2(\mathsf{pp}, \mathsf{s}, u_2, \mathsf{st}_2')$

Then, by Equations (7.7) and (7.8), for $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$, we have the following.

$$\begin{aligned} \left\{ \ \mathcal{S}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \ \right\} &\cong \left\{ \ \mathsf{st}_3 \ \middle| \ \begin{array}{l} (\mathsf{tr}_1, \mathsf{st}_2') \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \\ u_2 \leftarrow \varphi_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{tr}_1) \\ \mathsf{st}_3 \leftarrow \mathcal{S}_2(\mathsf{pp}, \mathsf{s}, u_2, \mathsf{st}_2') \end{array} \right\} \\ &\cong \left\{ \ \mathsf{st}_3 \ \middle| \ \begin{array}{l} ((\mathsf{tr}_1, \mathsf{st}_2'), w_2) \leftarrow \langle \mathcal{P}_1, \mathcal{V}_1^{**}(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \\ u_2 \leftarrow \varphi_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{tr}_1) \\ \mathsf{st}_3 \leftarrow \mathcal{S}_2(\mathsf{pp}, \mathsf{s}, u_2, \mathsf{st}_2') \end{array} \right\} \\ &\cong \left\{ \ \mathsf{st}_3 \ \middle| \ \begin{array}{l} ((\mathsf{tr}_1, \mathsf{st}_2'), w_2) \leftarrow \langle \mathcal{P}_1, \mathcal{V}_1^{**}(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \\ u_2 \leftarrow \varphi_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{tr}_1) \\ (\mathsf{st}_3, w_3) \leftarrow \langle \mathcal{P}_2, \mathcal{V}_2^{**}(\mathsf{st}_2') \rangle ((\mathsf{pk}, \mathsf{vk}), u_2, w_2) \end{array} \right\} \\ &\cong \left\{ \ \mathsf{st}_3 \ \middle| \ (\mathsf{st}_3, w_2) \leftarrow \langle \mathcal{P}, \mathcal{V}^* (\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \ \right\} \end{aligned}$$

Therefore, we have that $\Pi_2 \circ \Pi_1$ is zero-knowledge. □

Lemma 7.9 (Closure under Parallel Composition). (Honest-verifier) zero-knowledge is closed under parallel composition.

Proof. Consider zero-knowledge reductions of knowledge $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_1, \mathcal{V}_1)$ and $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_2, \mathcal{V}_2)$. Consider PPT $\mathcal{V}^*$ for $\Pi_1 \times \Pi_2$. We must construct a corresponding simulator EPT $\mathcal{S}$ for any PPT adversary $\mathcal{A}$.

Indeed, by construction, we have that $\mathcal{P}$ first runs $\mathcal{P}_1$ on statement and witness (u_1, w_1) to produce statement and witness (u_3, w_3). Then, $\mathcal{P}$ runs $\mathcal{P}_2$ on statement and witness (u_2, w_2) to produce statement and witness (u_4, w_4). As such, we have that $\mathcal{V}^*$ first runs some $\mathcal{V}_1^*$ (which takes the full input) and then runs some $\mathcal{V}_2^*$ such that $\mathcal{V}_1^*$ interacts with $\mathcal{P}_1$ and then passes some state to $\mathcal{V}_2^*$ which interacts with $\mathcal{P}_2$ before the two parties collectively produce the output $(\mathsf{st}_3, (w_3, w_4))$.

We construct PPT adversary $\mathcal{A}_1$ and $\mathcal{V}_1^{**}$ for Π_1 as follows

$\underline{\mathcal{A}_1(\mathsf{pp}) \to (\mathsf{s}, u_1, w_1, \mathsf{st}_1')}$:

1. Compute $(\mathsf{s}, (u_1, u_2), (w_1, w_2), \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$
2. Output $(\mathsf{s}, u_1, w_1, (\mathsf{st}_1', u_2))$

$\underline{\mathcal{V}_1^{**}(\mathsf{vk}, u_1, \mathsf{st}_1) \to \mathsf{st}_2}$:

1. Parse st_1 as (st_1', u_2)
2. Run $\mathcal{V}_1^*(\mathsf{vk}, (u_1, u_2), \mathsf{st}_1')$, which, at the end of interaction produces st_2.
3. Output st_2.

Then, by the zero-knowledge property of Π_1, we have that there exists EPT simulator $\mathcal{S}_1$ such that for $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$

$$\begin{aligned} &\left\{ \ \mathsf{st}_2 \ \middle| \ (\mathsf{st}_2, w_3) \leftarrow \langle \mathcal{P}_1, \mathcal{V}_1^{**}(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \ \right\} \cong \\ &\left\{ \ \mathsf{st}_2 \ \middle| \ \mathsf{st}_2 \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \ \right\}. \end{aligned} \tag{7.9}$$

Next, we construct PPT adversary $\mathcal{A}_2$ and $\mathcal{V}_2^{**}$ for Π_2 as follows.

$\underline{\mathcal{A}_2(\mathsf{pp}) \to (\mathsf{s}, u_2, w_2, \mathsf{st}_2)}$:

1. Compute $(\mathsf{s}, (u_1, u_2), (w_1, w_2), \mathsf{st}_1') \leftarrow \mathcal{A}(\mathsf{pp})$.
2. Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$
3. Compute $(\mathsf{st}_2, w_3) \leftarrow \langle \mathcal{P}_1, \mathcal{V}_1^{**}((\mathsf{st}_1', u_2)) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1)$.
4. Output $(\mathsf{s}, u_2, w_2, \mathsf{st}_2)$.

$\underline{\mathcal{V}_2^{**}(\mathsf{vk}, u_2, \mathsf{st}_2) \to \mathsf{st}_3}$:

1. Run $\mathcal{V}_2^*(\mathsf{vk}, u_2, \mathsf{st}_2)$, which, at the end of interaction produces st_3.
2. Output st_3.

By the zero-knowledge property of Π_2, we have that there exists EPT simulator $\mathcal{S}_2$ such that for $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, u_2, w_2, \mathsf{st}_2) \leftarrow \mathcal{A}_2(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$

$$\begin{aligned} &\left\{ \ \mathsf{st}_3 \ \middle| \ (\mathsf{st}_3, w_4) \leftarrow \langle \mathcal{P}_2, \mathcal{V}_2^{**}(\mathsf{st}_2) \rangle ((\mathsf{pk}, \mathsf{vk}), u_2, w_2) \ \right\} \cong \\ &\left\{ \ \mathsf{st}_3 \ \middle| \ \mathsf{st}_3 \leftarrow \mathcal{S}_2(\mathsf{pp}, \mathsf{s}, u_2, \mathsf{st}_2) \ \right\}. \end{aligned} \tag{7.10}$$

We construct a simulator $\mathcal{S}$ for $\Pi_1 \times \Pi_2$ as follows.

$\underline{\mathcal{S}(\mathsf{pp}, \mathsf{s}, (u_1, u_2), \mathsf{st}_1') \rightarrow \mathsf{st}_3}$:

1. Compute $\mathsf{st}_2 \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{s}, u_1, (\mathsf{st}_1', u_2))$
2. Compute and output $\mathsf{st}_3 \leftarrow \mathcal{S}_2(\mathsf{pp}, \mathsf{s}, u_2, \mathsf{st}_2)$

Then, by Equations (7.9) and (7.10) for $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, (u_1, u_2), (w_1, w_2), \mathsf{st}_1') \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1 \times \mathcal{R}_2$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$, we have the following.

$$\begin{aligned} \{\mathcal{S}(\mathsf{pp}, \mathsf{s}, (u_1, u_2), \mathsf{st}_1)\} &\cong \left\{ \ \mathsf{st}_3 \ \middle| \ \begin{array}{l} \mathsf{st}_2 \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{s}, u_1, (\mathsf{st}_1', u_2)) \\ \mathsf{st}_3 \leftarrow \mathcal{S}_2(\mathsf{pp}, \mathsf{s}, u_2, \mathsf{st}_2) \end{array} \ \right\} \\ &\cong \left\{ \ \mathsf{st}_3 \ \middle| \ \begin{array}{l} (\mathsf{st}_2, w_2) \leftarrow \langle \mathcal{P}_1, \mathcal{V}_1^{**}((\mathsf{st}_1', u_2)) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \\ \mathsf{st}_3 \leftarrow \mathcal{S}_2(\mathsf{pp}, \mathsf{s}, u_2, \mathsf{st}_2) \end{array} \ \right\} \\ &\cong \left\{ \ \mathsf{st}_3 \ \middle| \ \begin{array}{l} (\mathsf{st}_2, w_3) \leftarrow \langle \mathcal{P}_1, \mathcal{V}_1^{**}((\mathsf{st}_1', u_2)) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \\ (\mathsf{st}_3, w_4) \leftarrow \langle \mathcal{P}_2, \mathcal{V}_2^{**}(\mathsf{st}_2) \rangle ((\mathsf{pk}, \mathsf{vk}), u_2, w_2) \end{array} \ \right\} \\ &\cong \left\{ \ \mathsf{st}_3 \ \middle| \ (\mathsf{st}_3, (w_3, w_4)) \leftarrow \langle \mathcal{P}, \mathcal{V}^*(\mathsf{st}_1') \rangle ((\mathsf{pk}, \mathsf{vk}), (u_1, u_2), (w_1, w_2)) \ \right\} \end{aligned}$$

Therefore, we have that $\Pi_1 \times \Pi_2$ is zero-knowledge. □

Lemma 7.10 (Preservation of Strong Zero-Knowledge). Consider a reduction of knowledge $\Pi_2 : \mathcal{R}_2 \rightarrow \mathcal{R}_3$, and an (honest-verifier) strong zero-knowledge reduction of knowledge $\Pi_1 : \mathcal{R}_1 \rightarrow \mathcal{R}_2$. Then $\Pi_2 \circ \Pi_1$ is an (honest-verifier) strong zero-knowledge reduction of knowledge.

Proof. Let $\Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_1, \mathcal{V}_1)$ and $\Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_2, \mathcal{V}_2)$. Let $\Pi_2 \circ \Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$. Consider PPT $\mathcal{V}^*$ for $\Pi_2 \circ \Pi_1$. We must construct a corresponding EPT simulator $\mathcal{S}$ for any PPT adversary $\mathcal{A}$.

Indeed, by construction, we have that $\mathcal{P}$ first runs $\mathcal{P}_1$ on statement and witness (u_1, w_1) to produce statement and witness (u_2, w_2). Then $\mathcal{P}$ runs $\mathcal{P}_2$ on statement and witness (u_2, w_2) to produce statement and witness (u_3, w_3). As such, we have that $\mathcal{V}^*$ first runs some $\mathcal{V}_1^*$ and then runs some $\mathcal{V}_2^*$ such that $\mathcal{V}_1^*$ interacts with $\mathcal{P}_1$ and then passes some state to $\mathcal{V}_2^*$ which interacts with $\mathcal{P}_2$ before the two parties collectively produce the output (st_3, w_3).

We construct PPT adversary $\mathcal{A}_1$ and $\mathcal{V}_1^{**}$ for Π_1 as follows.

$\underline{\mathcal{A}_1(\mathsf{pp}) \rightarrow (\mathsf{s}, u_1, w_1, \mathsf{st}_1)}$:

1. Compute and output $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$.

$\underline{\mathcal{V}_1^{**}(\mathsf{vk}, u_1, \mathsf{st}_1) \to \mathsf{st}_2}$:

1. Run $\mathcal{V}_1^*(\mathsf{vk}, u_1, \mathsf{st}_1)$, which, at the end of interaction produces st_2'. Record the corresponding transcript as tr_1

2. Output $(\mathsf{tr}_1, \mathsf{st}_2')$.

Then, by the strong zero-knowledge property of Π_1, we have that there exists an EPT simulator $\mathcal{S}_1$ such that for $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$

$$\begin{aligned} &\left\{ \ (\mathsf{st}_2, w_2) \ \middle| \ (\mathsf{st}_2, w_2) \leftarrow \langle \mathcal{P}_1, \mathcal{V}_1^{**}(\mathsf{st}_1) \rangle((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \ \right\} \cong \\ &\left\{ \ (\mathsf{st}_2, w_2) \ \middle| \ (\mathsf{st}_2, w_2) \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \ \right\}. \end{aligned} \tag{7.11}$$

Given the simulator $\mathcal{S}_1$, we construct an EPT simulator $\mathcal{S}$ for $\Pi_2 \circ \Pi_1$ as follows. Let φ_1 be the deterministic functions guaranteed by the public reducibility of Π_1.

$\underline{\mathcal{S}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \to \mathsf{st}_3}$:

1. Compute $(\mathsf{st}_2, w_2) \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1)$

2. Parse st_2 as $(\mathsf{tr}_1, \mathsf{st}_2')$

3. Compute $u_2 \leftarrow \varphi_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{tr}_1)$

4. Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$

5. Compute and output $(\mathsf{st}_3, w_3) \leftarrow \langle \mathcal{P}_2, \mathcal{V}_2^*(\mathsf{st}_2') \rangle((\mathsf{pk}, \mathsf{vk}), u_2, w_2)$

Then, by Equation 7.11 for $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$, $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$, we have the following.

$$\begin{aligned} \left\{ \ \mathcal{S}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \ \right\} &\cong \left\{ \ (\mathsf{st}_3, w_3) \ \middle| \ \begin{array}{l} ((\mathsf{tr}_1, \mathsf{st}_2'), w_2) \leftarrow \mathcal{S}_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \\ u_2 \leftarrow \varphi_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{tr}_1) \\ (\mathsf{st}_3, w_3) \leftarrow \langle \mathcal{P}_2, \mathcal{V}_2^*(\mathsf{st}_2') \rangle((\mathsf{pk}, \mathsf{vk}), u_2, w_2) \end{array} \right\} \\ &\cong \left\{ \ (\mathsf{st}_3, w_3) \ \middle| \ \begin{array}{l} ((\mathsf{st}_2', \mathsf{tr}_1), w_2) \leftarrow \langle \mathcal{P}_1, \mathcal{V}_1^{**}(\mathsf{st}_1) \rangle((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \\ u_2 \leftarrow \varphi_1(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{tr}_1) \\ (\mathsf{st}_3, w_3) \leftarrow \langle \mathcal{P}_2, \mathcal{V}_2^*(\mathsf{st}_2') \rangle((\mathsf{pk}, \mathsf{vk}), u_2, w_2) \end{array} \right\} \\ &\cong \left\{ \ (\mathsf{st}_3, w_3) \ \middle| \ (\mathsf{st}_3, w_3) \leftarrow \langle \mathcal{P}, \mathcal{V}^*(\mathsf{st}_1) \rangle((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \ \right\} \end{aligned}$$

Therefore, $\Pi_2 \circ \Pi_1$ satisfies strong zero-knowledge. □

7.3.4 A Zero-Knowledge Transformation

We now present our zero-knowledge transformation.

Definition 7.7 (Blinding). A folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for $\mathcal{R}$ is blinding if there exists a *blinding distribution* $\mathcal{D}$ such that for any PPT adversary $\mathcal{V}^*$ there exists an EPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{A}$ given $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$, $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{pp})$ such that $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$ we have that

$$\left\{ (\mathsf{st}_2, w_2) \;\middle|\; \begin{array}{l} (u_{\mathsf{blind}}, w_{\mathsf{blind}}) \xleftarrow{\$} \mathcal{D}(\mathsf{pp}, \mathsf{s}) \\ (\mathsf{st}_2, w_2) \leftarrow \langle \mathcal{P}, \mathcal{V}^*(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), (u_1, u_{\mathsf{blind}}), (w_1, w_{\mathsf{blind}})) \end{array} \right\} \cong$$
$$\left\{ (\mathsf{st}_2, w_2) \;\middle|\; (\mathsf{st}_2, w_2) \leftarrow \mathcal{S}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \right\}.$$

A blinding folding scheme is called honest-verifier blinding if it satisfies blinding under an honest (but curious) verifier.

Lemma 7.11 (Closure under Parallel Composition). Consider an (honest-verifier) blinding folding scheme Π_1 for $\mathcal{R}_1$ and an (honest-verifier) blinding folding scheme Π_2 for $\mathcal{R}_2$. Then $\Pi_1 \times \Pi_2$ is an (honest-verifier) blinding zero-knowledge folding scheme for $\mathcal{R}_1 \times \mathcal{R}_2$.

Construction 7.6 (Zero-Knowledge Transformation). Consider a blinding folding scheme for $\mathcal{R}_1$, $\Pi_{\mathsf{fold}} = (\mathcal{G}, \mathcal{K}, \mathcal{P}_{\mathsf{fold}}, \mathcal{V}_{\mathsf{fold}})$, and reduction of knowledge $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \to \mathcal{R}_2$. We construct a strong zero-knowledge reduction of knowledge $\Pi' = (\mathcal{G}, \mathcal{K}, \mathcal{P}', \mathcal{V}') : \mathcal{R}_1 \to \mathcal{R}_2$ as follows.

$\underline{\mathcal{P}'(\mathsf{pk}, u_1, w_1) \to (u_2, w_2)}$:

1. Sample $(u_{\mathsf{blind}}, w_{\mathsf{blind}}) \in \mathcal{R}_1$ using the blinding distribution and send u_{blind} to the verifier.
2. Compute $(u_1', w_1') \leftarrow \mathcal{P}_{\mathsf{fold}}(\mathsf{pk}, (u_1, u_{\mathsf{blind}}), (w_1, w_{\mathsf{blind}}))$
3. Output $(u_2, w_2) \leftarrow \mathcal{P}(\mathsf{pk}, u_1', w_1')$

$\underline{\mathcal{V}'(\mathsf{vk}, u_1) \to u_2}$:

1. Receive u_{blind} from the prover
2. Compute $u_1' \leftarrow \mathcal{V}_{\mathsf{fold}}(\mathsf{vk}, (u_1, u_{\mathsf{blind}}))$
3. Output $u_2 \leftarrow \mathcal{V}(\mathsf{vk}, u_1')$

Theorem 7.4 (Zero-Knowledge Transformation). Construction 7.6 is a transformation from a blinding folding scheme for $\mathcal{R}_1$ and a reduction of knowledge of type $\mathcal{R}_1 \to \mathcal{R}_2$ to a strong zero-knowledge reduction of knowledge of type $\mathcal{R}_1 \to \mathcal{R}_2$.

Proof. Completeness and knowledge-soundness follow from the completeness and knowledge-soundness of Π_{fold} and Π and the closure of completeness and knowledge soundness under sequential composition (Theorem 2.1). Strong zero-knowledge follows from Lemma 7.12. □

Lemma 7.12 (Zero-Knowledge). Construction 7.6 satisfies strong zero-knowledge.

Proof. By construction, we that $\Pi' = \Pi \circ \Pi_{\mathsf{blindfold}}$ where $\Pi_{\mathsf{blindfold}}$ is the first two steps of the prover and verifier of Π'. By the preservation of strong zero-knowledge under sequential composition (Lemma 7.10), it is sufficient to demonstrate that $\Pi_{\mathsf{blindfold}}$ has strong zero-knowledge to show that Π' has strong zero-knowledge.

Let $\mathcal{P}_{\mathsf{blindfold}}$ and $\mathcal{V}_{\mathsf{blindfold}}$ be the prover and verifier associated with $\Pi_{\mathsf{blindfold}}$. Let $\mathcal{P}_{\mathsf{blindfold}}$ be further decomposed into $\mathcal{P}_{\mathsf{blind}}$, which is first step in which $\mathcal{P}_{\mathsf{blindfold}}$ samples and sends u_{blind} and $\mathcal{P}_{\mathsf{fold}}$. Consider PPT adversaries $\mathcal{A}$ and $\mathcal{V}^*_{\mathsf{blindfold}}$ for $\Pi_{\mathsf{blindfold}}$. By construction, we have that $\mathcal{V}^*_{\mathsf{blindfold}}$ first runs some $\mathcal{V}_{\mathsf{blind}}$ which interact with $\mathcal{P}_{\mathsf{blind}}$, which then passes some state $\mathsf{st}_{\mathsf{blind}}$ into some $\mathcal{V}^*_{\mathsf{fold}}$. Then, for $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$, $(\mathsf{s}, u_1, w_1, \mathsf{st}_1) \in \mathcal{R}_1 \leftarrow \mathcal{A}(\mathsf{pp})$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s})$, we have that the following two distributions are equal

$$\left\{ \ (\mathsf{st}'_1, w'_1) \ \middle| \ (\mathsf{st}'_1, w'_1) \leftarrow \langle \mathcal{P}_{\mathsf{blindfold}}, \mathcal{V}^*_{\mathsf{blindfold}}(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), u_1, w_1) \ \right\} \tag{7.12}$$

and

$$\left\{ (\mathsf{st}'_1, w'_1) \ \middle| \ \begin{array}{l} (u_{\mathsf{blind}}, w_{\mathsf{blind}}) \leftarrow \mathcal{D}(\mathsf{pp}, \mathsf{s}), \\ \mathsf{st}_{\mathsf{blind}} \leftarrow \mathcal{V}^*_{\mathsf{blind}}(\mathsf{st}_1, u_{\mathsf{blind}}), \\ (\mathsf{st}'_1, w'_1) \leftarrow \langle \mathcal{P}_{\mathsf{fold}}, \mathcal{V}^*_{\mathsf{fold}}(\mathsf{st}_{\mathsf{blind}}) \rangle ((\mathsf{pk}, \mathsf{vk}), (u_1, u_{\mathsf{blind}}), (w_1, w_{\mathsf{blind}})) \end{array} \right\}. \tag{7.13}$$

Then, we define $\mathcal{V}^*$ as follows: On input $(\mathsf{st}_1, u_{\mathsf{blind}})$, first compute $\mathsf{st}_{\mathsf{blind}} \leftarrow \mathcal{V}^*_{\mathsf{blind}}(\mathsf{st}_1, u_{\mathsf{blind}})$ and then run $\mathcal{V}^*_{\mathsf{fold}}(\mathsf{st}_{\mathsf{blind}})$. Then, by construction we have that Distribution 7.13 is equal to

$$\left\{ (\mathsf{st}'_1, w'_1) \ \middle| \ \begin{array}{l} (u_{\mathsf{blind}}, w_{\mathsf{blind}}) \leftarrow \mathcal{D}(\mathsf{pp}, \mathsf{s}), \\ (\mathsf{st}'_1, w'_1) \leftarrow \langle \mathcal{P}_{\mathsf{fold}}, \mathcal{V}^*(\mathsf{st}_1) \rangle ((\mathsf{pk}, \mathsf{vk}), (u_1, u_{\mathsf{blind}}), (w_1, w_{\mathsf{blind}})) \end{array} \right\}. \tag{7.14}$$

Then, by the blinding property of Π_{fold}, there exists an EPT simulator such that Distribution 7.14 is equal to

$$\left\{ \ (\mathsf{st}'_1, w'_1) \ \middle| \ (\mathsf{st}'_1, w'_1) \leftarrow \mathcal{S}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st}_1) \ \right\}. \tag{7.15}$$

Therefore, we have that $\Pi_{\mathsf{blindfold}}$ has strong zero-knowledge, which implies that Π' has strong zero-knowledge. □

Corollary 7.1 (Strong Zero-Knowledge Self-Reduction). A blinding folding scheme for $\mathcal{R}$ implies a strong zero-knowledge reduction of knowledge from $\mathcal{R}$ to $\mathcal{R}$.

7.3.5 Applications

In this section, we begin by applying our zero-knowledge transformation to linear relations.

Example 7.1 (Zero-Knowledge Homomorphism Preimage Argument [35]). Consider group $\mathbb{G}$ and corresponding scalar field $\mathbb{F}$. Consider homomorphism $G \in \mathbb{G}^n$. We define the homomorphism preimage relation as follows.

$$\mathcal{R}_{\mathsf{HPI}}(G) = \left\{ \ (\overline{A}, A) \in (\mathbb{G}, \mathbb{F}^n) \ \middle| \ \langle G, A \rangle = \overline{A} \ \right\}.$$

where $\langle G, A \rangle$ denotes the inner-product of G and A.

An instance-witness pair in $\mathcal{R}_{\mathsf{HPI}}(G)$ is randomly sampled by first randomly sampling the witness $A_{\mathsf{blind}} \xleftarrow{\$} \mathbb{F}^n$ and then computing $\overline{A}_{\mathsf{blind}} \leftarrow \langle G, A_{\mathsf{blind}} \rangle$. We construct a blinding folding scheme for $\mathcal{R}_{\mathsf{HPI}}(G)$ as follows.

1. The prover and verifier are provided input instances $\overline{A} \in \mathbb{G}$ and $\overline{A}_{\mathsf{blind}} \in \mathbb{G}$. The prover is additionally provided with the corresponding witnesses $A \in \mathbb{F}^n$ and $A_{\mathsf{blind}} \in \mathbb{F}^n$.

2. The verifier begins by sending a random challenge $r \xleftarrow{\$} \mathbb{F}$.

3. Together, the prover and verifier compute and output the folded instance $\overline{A}' \leftarrow \overline{A}_{\mathsf{blind}} + r \cdot \overline{A}$. Privately, the prover computes and outputs the folded witness $A' \leftarrow A_{\mathsf{blind}} + r \cdot A$

To prove that this folding scheme is honest-verifier blinding, we must show that a simulator can simulate the joint distribution of the prover's folded witness and an honest (but curious) verifier's output. Indeed, the simulator can randomly sample $(\overline{A}', A') \xleftarrow{\$} \mathcal{R}_{\mathsf{HPI}}$ and $r \xleftarrow{\$} \mathbb{F}$. The simulator simulates $\overline{A}_{\mathsf{blind}} \leftarrow \overline{A}' - r \cdot \overline{A}$. The simulator then computes the honest-but-curious verifier's output st by providing input $(\overline{A}, \overline{A}_{\mathsf{blind}})$ and randomness r. The simulator produces the simulated output (A', st), which is indistinguishable from an honestly produced output.

Then, by Theorem 7.4 any proof of knowledge Π for $\mathcal{R}_{\mathsf{HPI}}$ can be transformed into a zero-knowledge proof of knowledge for $\mathcal{R}_{\mathsf{HPI}}$ by prepending the blinding folding scheme. The original proof of knowledge Π might, for instance, trivially reveal A' to the verifier. Alternatively, to achieve a logarithmic communication overhead, Π may recursively reduce the the size of the witness until $A' \in \mathbb{F}$ [36].

Definition 7.8 (Tensor Relation). For vector spaces T, W and V over field $\mathbb{F}$, such that $T \cong \hom(W, V)$ we define the tensor relation for T over structure, instance, witness pairs as follows

$$\mathcal{R}_{\mathsf{tensor}}(T) = \left\{ \ (t, v, w) \ \middle| \ \begin{array}{l} t \in T, v \in V, w \in W, \\ u(w) = v \end{array} \ \right\}$$

Construction 7.7 (A Blinding Folding Scheme for the Tensor Relation). Consider vector spaces T, W and V over field $\mathbb{F}$, such that $T \cong \hom(W, V)$. We construct a blinding folding scheme for $\mathcal{R}_{\mathsf{tensor}}(T)$.

Given structure $t \in T$, an instance-witness pair $(v_{\mathsf{blind}}, w_{\mathsf{blind}}) \in \mathcal{R}_{\mathsf{tensor}}(T)$ is randomly sampled by first randomly sampling $w_{\mathsf{blind}} \xleftarrow{\$} W$ and then computing $v_{\mathsf{blind}} \leftarrow t(w_{\mathsf{blind}})$.

As there is no notion of public parameter, the generator is defined to return the empty string on all inputs. The encoder is defined to set the prover key and verifier key to the input structure $t \in T$. We define the prover and verifier as follows.

$\underline{\mathcal{P}'(t, (v, v_{\mathsf{blind}}), (w, w_{\mathsf{blind}})) \to (v', w')}$:

1. Receive challenge $r \in \mathbb{F}$ from the verifier
2. Output $(v', w') \leftarrow (w_{\mathsf{blind}} + r \cdot w, v_{\mathsf{blind}} + r \cdot v)$

$\underline{\mathcal{V}'(t, (v, v_{\mathsf{blind}})) \to v'}$:

1. Send random challenge $r \xleftarrow{\$} \mathbb{F}$ to the prover
2. Output $v' \leftarrow v_{\mathsf{blind}} + r \cdot v$

Lemma 7.13 (A Blinding Folding Scheme for the Tensor Relation). Consider vector spaces T, W and V over field $\mathbb{F}$, such that $T \cong \hom(W, V)$. Construction 7.7 is an honest-verifier blinding folding scheme for $\mathcal{R}_{\mathsf{tensor}}(T)$.

Proof. Completeness and knowledge-soundness follow from observation. Blinding follows from the same reasoning as Example 7.1. □

Definition 7.9 (Committed Sumcheck Relation). Consider size parameters $n \in \mathbb{N}$ and $N = 2^\ell$. Consider group $\mathbb{G}$ and corresponding scalar field $\mathbb{F}$. We define the *committed* sumcheck relationas follows.

$$\mathcal{R}_{\mathsf{CSC}} = \left\{ (G, (\overline{P}, \sigma), P) \in (\mathbb{G}^N, (\mathbb{G}, \mathbb{F}), \mathbb{F}[X_1, \ldots, X_n]) \;\middle|\; \begin{array}{l} \overline{P} = G(P) \\ \sigma = \sum_{x_1, \ldots, x_n \in \{0,1\}} P(x_1, \ldots, x_n) \end{array} \right\}.$$

where $G(P)$ denotes the inner-product of G and the coefficients of P.

Example 7.2 (A Blinding Folding Scheme for the Committed Sumcheck Relation [53]). Suppose P has degree bound d in each variable. Consider the tensor of evaluation points

$$X = \sum_{x_1, \ldots, x_n \in \{0,1\}} \bigotimes_{i \in [n]} \bigoplus_{j \in \{0, \ldots, d\}} x_i^j.$$

Then, as discussed in Section 3.2, for $P \in \hom(\mathbb{F}^n, \mathbb{F})$, we have that

$$\sigma = \sum_{x_1, \ldots, x_n \in \{0,1\}} P(x_1, \ldots, x_n)$$

if and only if $\sigma = X(P)$. Moreover, have that $G \in \mathbb{G}^N \cong \hom(\mathbb{F}^N, \mathbb{G})$. Then, we have that $(G, (\overline{P}, \sigma), P) \in \mathcal{R}_{\mathsf{CSC}}$ if and only if

$$(G \oplus X, (\overline{P}, \sigma), P) \in \mathcal{R}_{\mathsf{tensor}}(\hom(\mathbb{F}^n, \mathbb{G}) \oplus \hom(\mathbb{F}^n, \mathbb{F}))$$

As such, the blinding folding scheme for the tensor relation directly enables a blinding folding scheme for the committed sumcheck relation.

Example 7.3 (A Blinding Folding Scheme for Polynomial Evaluation [35]). Consider the polynomial evaluation relation (Definition 4.10) instantiated over the Pedersen commitment scheme. Consider polynomial evaluation instance-witness pair

$$(G, (\overline{P}, x, y), P) \in \mathcal{R}_{\mathsf{polyeval}}.$$

Now, consider the expanded evaluation point

$$X = \bigotimes_{i \in [n]} \bigoplus_{j \in \{0,\ldots,d\}} x_i^j.$$

Then, we have that if and only if

$$(G \oplus X, (\overline{P}, y), P) \in \mathcal{R}_{\mathsf{tensor}}(\hom(\mathbb{F}^n, \mathbb{G}), \hom(\mathbb{F}^n, \mathbb{F}))$$

Thus, the blinding folding scheme for the tensor relation directly enables a blinding folding scheme for the polynomial evaluation relation over the Pedersen commitment scheme.

We now apply our zero-knowledge transformation to a non-linear relation, committed relaxed R1CS.

Lemma 7.14 (A Blinding Folding Scheme for Committed Relaxed R1CS). The folding scheme for committed relaxed R1CS is (Construction 4.1) is an honest-verifier blinding folding scheme.

Proof. Consider arbitrary PPT adversary $\mathcal{A}$ and honest-but-curious verifier $\mathcal{V}^*$. To prove blinding, we construct an EPT simulator $\mathcal{S}$ which simulates the joint distribution of the verifier's output and the prover's output witness. as follows.

$\underline{\mathcal{S}(\mathsf{pp}, (A, B, C), (\overline{E}, u, \overline{W}, \mathsf{x}), \mathsf{st}) \to (\mathsf{st}', (E', r_{E'}, W', r_{W'}))}$:

1. Sample the input blinding statement-witness pair $(\overline{E}_{\mathsf{blind}}, u_{\mathsf{blind}}, \overline{W}_{\mathsf{blind}}, \mathsf{x}_{\mathsf{blind}})$ according to the blinding distribution
2. Sample the folded statement-witness pair $((\overline{E}', u', \overline{W}', \mathsf{x}'), (E', r_{E'}, W', r_{W'}))$ according to the blinding distribution
3. Sample the verifier's challenge $r \xleftarrow{\$} \mathbb{F}$
4. Solve for the prover's first message $\overline{T} \leftarrow r^{-1} \cdot (\overline{E}' - \overline{E} - r^2 \cdot \overline{E}_{\mathsf{blind}})$
5. Compute the verifier key $\mathsf{vk} \leftarrow \mathcal{K}(\mathsf{pp}, (A, B, C))$
6. Run the verifier $\mathcal{V}^*$ on input vk, instances $(\overline{E}, u, \overline{W}, \mathsf{x})$ and $(\overline{E}_{\mathsf{blind}}, u_{\mathsf{blind}}, \overline{W}_{\mathsf{blind}}, \mathsf{x}_{\mathsf{blind}})$, and st. Let st' be the the output of $\mathcal{V}^*$. Instantiate the verifier randomness to r and send the first message $\overline{T}$.
7. Output st' and $(E', r_{E'}, W', r_{W'})$

We now argue that the simulator (i.e. ideal setting) produces an output that is indistinguishable from the prover and verifier output (i.e. the real setting). Indeed, suppose that $\mathsf{pp} \leftarrow \mathcal{G}(\lambda)$, $((A, B, C), (\overline{E}, u, \overline{W}, \mathsf{x}), (E, r_E, W, r_W), \mathsf{st}) \leftarrow \mathcal{A}$, and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, (A, B, C))$.

In both the real and ideal setting $(\overline{E}_{\mathsf{blind}}, u_{\mathsf{blind}}, \overline{W}_{\mathsf{blind}}, \mathsf{x}_{\mathsf{blind}})$ are randomly sampled from the blinding distribution and therefore is indistinguishable.

In the honest setting, by construction, we have that

$$\begin{aligned} W' &\leftarrow W + r \cdot W_{\mathsf{blind}} \\ \mathsf{x}' &\leftarrow \mathsf{x} + r \cdot \mathsf{x}_{\mathsf{blind}} \\ u' &\leftarrow u + r \cdot u_{\mathsf{blind}} \end{aligned}$$

Because W_{blind}, $\mathsf{x}_{\mathsf{blind}}$, and u_{blind} are uniformly random, we have that W', x', and u' are uniformly random. Moreover, r'_E and r'_W are computed as follows.

$$\begin{aligned} r'_E &\leftarrow r_E + r \cdot r_T + r^2 \cdot r_{E_{\mathsf{blind}}} \\ r'_W &\leftarrow r_W + r \cdot r_{W_{\mathsf{blind}}} \end{aligned}$$

By the same argument as above, we have that r'_E and r'_W are uniformly random.

Then, we have that E', $\overline{E}'$, and $\overline{W}'$ are completely determined by the prior values. Therefore, because W', x', u', r'_E and r'_W are also randomly sampled in the idealized setting. we have that the folded instance witness pair $((\overline{E}', u', \overline{W}', \mathsf{x}'), (E', r_{E'}, W', r_{W'}))$ is indistinguishable in both the real and ideal setting.

Moreover, because the verifier $\mathcal{V}^*$ interacts honestly with the prover, the challenge r sampled by $\mathcal{V}^*$ in the real setting is indistinguishable from the challenge r sampled by $\mathcal{S}$ in the ideal setting.

Then, because $\overline{T}$ is completely determined by prior values in both the real and ideal setting, we have that $\overline{T}$ is indistinguishable in both the real and ideal setting.

This implies that the view of the verifier $\mathcal{V}^*$ is indistinguishable in both the real and ideal setting. Therefore, the output st' is indistinguishable in both the real and ideal setting.

Putting together the prior assertions, we have that the simulator output

$$(\mathsf{st}', (E', r_{E'}, W', r_{W'}))$$

is indistinguishable from that of the interaction between the honest prover and $\mathcal{V}^*$. Therefore, honest-verifier blinding holds. □

Chapter 8

The Category of Proofs of Knowledge

Premature abstraction falls on deaf ears.

– Morris Kline,
Mathematical Thought from Ancient to Modern Times

Much like how group theory can be viewed as the general theory of anything that can be *added* (e.g., elliptic curve points), and ring theory as the theory of anything that can be *added and multiplied* (e.g., polynomials), *category theory* can be viewed as the general theory of anything that can be *composed*. As composition is the central concept in many domains of mathematics, category theory is used today as a unifying framework for large swathes of mathematics. Our aim in this chapter is to understand the higher-level structural properties of proofs of knowledge through the lens of category theory. Armed with the categorical language, we will provide a mathematical account of the statement that *proofs of knowledge are maps between propositions of knowledge*.

To understand any mathematical domain categorically, we must first specify the particular object in question. Next, we must specify an appropriate notion of transformations or maps over these objects. For example, in linear algebra, the objects are vectors that are transformed via matrices. In abstract algebra, the objects are groups, rings, and fields that are are transformed via homomorphisms. In theoretical computer science, the objects are computational problems (expressed as languages or relations) that are mapped via reductions. We must additionally specify how these maps are composed. For instance, matrices are composed with matrix multiplication, homomorphisms are composed by functional

composition, and reductions are composed with subroutine substitution.

A category is essentially determined by a class of objects, transformations over these objects, and a composition operator over these transformations. Most mathematical objects often feature additional structure. For instance, any two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ induce a new group $\mathbb{G}$ which consists of the Cartesian products of elements in groups $\mathbb{G}_1$ and $\mathbb{G}_2$. Thus, groups form a *monoidal* category, which supports Cartesian products among the elements. One of the central benefits of the categorical lens is that many disparate mathematical objects can be considered equivalent at the categorical level. Thus, we can generically define constructions on say monoidal categories, and initialize these constructions in any mathematical object formalized as a monoidal category.

Our goal is to extend these benefits to the domain of theoretical computer science by understanding reductions categorically. Indeed, we can envision a category where objects are search problems (in NP) and morphisms between these objects are, say, Cook reductions. If we would like to be more strict with the permissible reductions we can instead study the subcategory of Karp reductions. As both Cook reductions and Karp reductions are well understood, it's not immediately clear what a categorical approach affords. Alternatively, as reductions of knowledge are a nascent concept, studying corresponding category offers a promising route to better understanding how these objects can be composed and transformed.

The first benefit is that we can study reductions at an appropriate level of abstraction and apply generic constructions in category theory to these reductions. In particular, we show that the category of reductions of knowledge form a *symmetric monoidal category*, which is the category that captures linear logics and linear type systems. Thus, all prior results regarding symmetric monoidal categories apply to reductions of knowledge.

The second benefit is that category theory offers a powerful language for organizing various techniques in the proofs of knowledge literature. For instance, we can view non-interactive reductions or public-coin reductions as a subcategory of the category of reductions of knowledge. Then, we can view generic transformations over reductions, such as the (weak) Fiat-Shamir transformation (Construction 8.2), which takes a interactive reduction and produces a non-interactive reduction, as functors. In particular, the Fiat-Shamir transformation is a homomorphism from the category of public-coin reductions of knowledge to the category of non-interactive reductions of knowledge in the random oracle model.

As a preliminary testament to the merits of a categorical approach, we demonstrate that a reduction of knowledge of type $\mathcal{R}_1 \to \mathcal{R}_2$ can be understood as a (linear) transformation from a proof of knowledge for $\mathcal{R}_2$ to a proof of knowledge for $\mathcal{R}_1$. This demonstrates that every reduction of knowledge induces a corresponding compiler over proofs of knowledge, and that every *linear* compiler over proofs of knowledge induces a corresponding reduction of knowledge.

8.1 Overview of Category Theory

In this section, we define the necessary concepts in category theory. We start by formally defining a category, which, like a group, or ring, or field serves as a basic mathematical abstraction for which we can fill in concrete objects of interest. Unlike groups, rings, or fields however, a category is intentionally left with minimal requirements. This enables a large variety of mathematical objects to be viewed as particular types of categories. In particular, a category is only determined by a set of objects, maps over these objects, and a coherent composition operator over these maps. As a example, reductions of knowledge can be composed sequentially and thus form a category when interpreted as maps over relations.

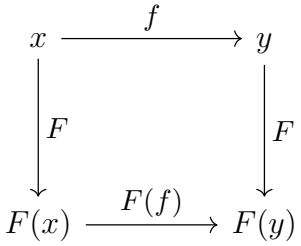
Definition 8.1 (Category). A category $\mathbb{C}$ consists of a class of objects and sets of morphisms between two objects. Each morphism is associated with domain object and a codomain object. We will write $f : A \to B$ to indicate that morphism f has domain A and codomain B. There exists a composition operation, denoted $\circ$, such that for any objects A, B, C, and for any morphisms $f : A \to B$ and $g \in B \to C$, we have that $g \circ f : A \to C$. The objects and morphisms satisfy the following axioms

(i) Associativity: For all $f : A \to B$, $g : B \to C$, $h : C \to D$, we have that $h \circ (g \circ f) = (h \circ g) \circ f$.

(ii) Identity: For all objects B, there exists a morphism $1_B : B \to B$, such that for all morphisms $f : A \to B$, $g : B \to C$ we have that $1_B \circ f = f$ and $g \circ 1_B = g$.

Just as groups, rings, and fields have homomorphisms which map from one space to another, categories too can have a notion of a homomorphism that maps from one category to another. Such homomorphisms are called functors. In particular, a functor $F : \mathbb{C} \to \mathbb{D}$ must be such that composing two maps in $\mathbb{C}$ and then applying F should produce the same result as applying F to each map and composing in $\mathbb{D}$. As an example, the (weak) Fiat-Shamir transformation can be viewed as a functor from the category of public-coin reductions of knowledge to the category of non-interactive reductions of knowledge in the random oracle model.

Definition 8.2 (Functor). A functor F from a category $\mathbb{C}$ to category $\mathbb{D}$ (denoted $F : \mathbb{C} \to \mathbb{D}$) is a map sending each object $x \in \mathbb{C}$ to an object $F(x) \in \mathbb{D}$ and each morphism $f : x \to y$ to morphism $F(f) : F(x) \to F(y)$ in $\mathbb{D}$. That is, the following diagram commutes.



Moreover, for morphisms g, f in and object x in $\mathbb{C}$, we must have that

(i) $F(g \circ f) = F(g) \circ F(f)$

(ii) $F(1_x) = 1_{F(x)}$

We have that functors can be composed naturally.

Lemma 8.1 (Functor Composition). Given functors $F : \mathbb{A} \to \mathbb{B}$ and $G : \mathbb{B} \to \mathbb{C}$ then $G \circ F$ is a functor from $\mathbb{A}$ to $\mathbb{C}$.

One of the core affordances of the categorical approach is that maps themselves can be treated as objects. Then, we can consider *higher-order* maps that transform these maps. Naturally then, we can consider functors themselves as objects, and consider the homomorphisms over these objects. Such homomorphisms are called natural transformations. Intuitively, if functors $F : \mathbb{C} \to \mathbb{D}$ and $G : \mathbb{C} \to \mathbb{D}$ can be viewed as "wrapping" an object (and morphism) in $\mathbb{C}$ to produce an object in $\mathbb{D}$, then a natural transformation $\eta : F \to G$, can be viewed as transforming one wrapped object to another wrapped object in a way that preserves the underlying object.

As an example, consider the weak Fiat-Shamir functor F_{FS} which transforms a public-coin reduction to a non-interactive reduction in the random oracle model. Now consider the Fiat-Shamir *heuristic* functor F_{FSH}, which, instead of using a random oracle, directly uses a cryptographic hash function (as was originally presented by Fiat and Shamir). As such, F_{FSH} (heuristically) maps from the category of public-coin reductions to the standard category of reductions (that is, in the plain model). Now consider an arbitrary public-coin reduction Π. Then, the reduction $F_{\mathsf{FS}}(\Pi)$ can be naturally transformed into the reduction $F_{\mathsf{FSH}}(\Pi)$ without touching Π by instantiating the random oracle with a cryptographic hash function. As such, we say that the natural transformation $\eta : F_{\mathsf{FS}} \to F_{\mathsf{FSH}}$ is *natural* (that is, polymorphic) in the underlying reduction Π.

We define natural transformations formally as follows.

Definition 8.3 (Natural Transformation). Given categories $\mathbb{C}$ and $\mathbb{D}$ and functors $F, G : \mathbb{C} \to \mathbb{D}$, a natural transformation $\eta : F \to G$ between them is an assignment to every object x in $\mathbb{C}$ of a morphism $\eta_x : F(x) \to G(x)$ in $\mathbb{D}$ (called the component of η) such that for any morphism $f : x \to y$ in $\mathbb{G}$ we have that the following diagram commutes.

$$\begin{array}{ccc} F(x) & \xrightarrow{\eta_x} & G(x) \\ \Big\downarrow{\scriptstyle F(f)} & & \Big\downarrow{\scriptstyle G(f)} \\ F(y) & \xrightarrow{\eta_y} & G(y) \end{array}$$

If each component η_x is an isomorphism in $\mathbb{D}$, then we say that η is a natural isomorphism.

While reductions of knowledge form an ambient category, we know that they possess additional structure. In particular, given reductions $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 : \mathcal{R}_3 \to \mathcal{R}_3$, we have that $\Pi_1 \times \Pi_2 : \mathcal{R}_1 \times \mathcal{R}_3 \to \mathcal{R}_2 \times \mathcal{R}_4$ is a reduction of knowledge in the same

category. This additional structure is precisely captured by a *monoidal category* which comes equipped with a monoidal product $\otimes$ between objects and maps that supports natural coherence conditions.

Definition 8.4 (Monoidal Category). A monoidal category is a category $\mathbb{C}$ equipped with

(i) a functor $\otimes : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$,

(ii) an object I, called the monoidal unit,

(iii) a natural isomorphism, $\alpha : (A \otimes B) \otimes C \cong A \otimes (B \otimes C)$, that is natural in A, B, and C called an associator,

(iv) a natural isomorphism, $\lambda : (I \otimes A) \cong A$, that is natural in A called the left unitor, and

(v) a natural isomorphism, $\rho : (A \otimes I) \cong A$, that is natural in A called the right unitor,

such that

(i) $(1_A \otimes \lambda_B) \circ \alpha_{A,I,B} = \rho_A \otimes 1_B$, and

(ii) $\alpha_{A\otimes B,C,D} \circ \alpha_{A,B,C\otimes D} = (\alpha_{A,B,C} \otimes 1_D) \circ (\alpha_{A,B\otimes C,D}) \circ (1_A \otimes \alpha_{B,C,D})$.

Reductions of knowledge form more than just a monoidal category however. Observe that $\mathcal{R}_1 \times \mathcal{R}_2$ is essentially the same relation as $\mathcal{R}_2 \times \mathcal{R}_1$ and $\Pi_1 \times \Pi_2$ is essentially the same reduction as $\Pi_2 \times \Pi_1$. This commutativity property is captured by a *symmetric monoidal category*.

Definition 8.5 (Symmetric Monoidal Category). A symmetric monoidal category is a monoidal category equipped with a natural isomorphism $s : (A \otimes B) \cong (B \otimes A)$ natural in A and B called the swap map such that

$$\rho_A = \lambda_A \circ s_{A,I}$$

and

$$(1_B \otimes s_{A,C}) \circ (\alpha_{B,A,C}) \circ (s_{A,B} \otimes 1_C) = (\alpha_{B,C,A}) \circ (s_{A,B\otimes C}) \circ (\alpha_{A,B,C}).$$

8.2 The Category of Reductions of Knowledge

In this section, we formally demonstrate that reductions of knowledge form a symmetric monoidal category.

Construction 8.1 (The Category of Reductions of Knowledge, RoK). We first define the components needed to determine an ambient category RoK.

- *Objects:* Objects are determined by relations (denoted $\mathcal{R}_1, \mathcal{R}_2, \ldots$) in NP. An object determined by relation $\mathcal{R}_1$ is defined as the set of instance-witness pairs in $\mathcal{R}_1$.

- *Morphisms:* Morphisms from relation $\mathcal{R}_1$ to relation $\mathcal{R}_2$ are defined to be reductions of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$.
- *Identity:* For relation $\mathcal{R}$, we define the identity morphism $1_{\mathcal{R}} : \mathcal{R} \to \mathcal{R}$ as the reduction of knowledge in which the prover and verifier output their input statement and witness. In particular we define $1_{\mathcal{R}} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ where $\mathcal{G}$ and $\mathcal{K}$ are defined arbitrarily and $\mathcal{P}$ and $\mathcal{V}$ are defined as follows.

$$\mathcal{P}(\mathsf{pk}, u_1, w_1) = (u_1, w_1)$$
$$\mathcal{V}(\mathsf{vk}, u_1) = u_1.$$

- *Composition:* The composition operator $\circ$ is defined to be the sequential composition operator as described in Theorem 2.1.

We now define the monoidal product operation to show that the RoK forms a symmetric monoidal category.

- *Monoidal Product:* We define the functor $\otimes : \mathsf{RoK} \times \mathsf{RoK} \to \mathsf{RoK}$ as follows. Given relations $\mathcal{R}_1$ and $\mathcal{R}_2$ we define $\mathcal{R}_1 \otimes \mathcal{R}_2$ as the relation $\mathcal{R}_1 \times \mathcal{R}_2$. Given reductions of knowledge Π_1 and Π_2 we define $\Pi_1 \otimes \Pi_2$ as $\Pi_1 \times \Pi_2$.
- *Monoidal Unit:* We define the identity object as the singleton relation $\mathcal{R}_I = \{(\bot, \bot)\}$.

Theorem 8.1 (The Category of Reductions of Knowledge). RoK is a symmetric monoidal category.

Proof. We first demonstrate that RoK forms a category. Indeed, for any reductions of knowledge $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$, and $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3$ we have that $\Pi_2 \circ \Pi_1 : \mathcal{R}_1 \to \mathcal{R}_3$ by Theorem 2.1. Moreover, for $\Pi_3 : \mathcal{R}_3 \to \mathcal{R}_4$, by construction, we have that $\Pi_3 \circ (\Pi_2 \circ \Pi_1) = (\Pi_3 \circ \Pi_2) \circ \Pi_1$. Additionally, by construction, we have that $1_{\mathcal{R}_2} \circ \Pi_2 = \Pi_2$ and $\Pi_1 \circ 1_{\mathcal{R}_1} = \Pi_1$.

Next, we argue that RoK forms a monoidal category. Indeed, consider arbitrary reductions of knowledge Π_1, Π_2, Π_3 and Π_4. We define the components of the associator $\alpha_{\mathcal{R}_1,\mathcal{R}_2,\mathcal{R}_3} : (\mathcal{R}_1 \otimes \mathcal{R}_2) \otimes \mathcal{R}_3 \cong \mathcal{R}_1 \otimes (\mathcal{R}_2 \otimes \mathcal{R}_3)$ as follows

$$\alpha_{\mathcal{R}_1,\mathcal{R}_2,\mathcal{R}_3}((((u_1, u_2), u_3), ((w_1, w_2), w_3))) = ((u_1, (u_2, u_3)), (w_1, (w_2, w_3))).$$

By observation, we have that α is a natural isomorphism. We define the components of the left unitor $\lambda_{\mathcal{R}} : (\mathcal{R}_I \otimes \mathcal{R}) \cong \mathcal{R}$ as

$$\lambda_{\mathcal{R}}((\bot, u), (\bot, w)) = (u, w).$$

Likewise, we define the components of the right unitor $\rho_{\mathcal{R}} : (\mathcal{R} \otimes \mathcal{R}_I) \otimes \mathcal{R}$ as

$$\rho_{\mathcal{R}}((u, \bot), (w, \bot)) = (u, w).$$

By observation, we have that λ and ρ are natural isomorphisms. Moreover, by observation, α, λ, and ρ satisfy the required coherence conditions.

Next, we argue that RoK forms a symmetric monoidal category. We define the components of the swap map s as follows

$$s_{\mathcal{R}_1,\mathcal{R}_2}((u_1,u_2),(w_1,w_2)) = ((u_2,u_1),(w_2,w_1)).$$

By observation we have that s is a natural isomorphism. Moreover, by observation, we have that s follows the required coherence condition. □

8.3 Transformations as Functors

In this section, we demonstrate that we can interpret particular transformations over reductions of knowledge as functors from one subcategory (e.g., the subcategory of public-coin reductions) to another. Recall that a functor F is required to satisfy a strong homomorphism property: Composing two reductions and then applying F should be equivalent to applying F to each reduction individually, and then composing. Unfortunately, the transformations we have discussed thus far (Chapter 7) do *not* satisfy this strict condition. For instance, consider the zero-knowledge transformation (Section 7.3), which randomizes the initial instance-witness pair using a blinding folding scheme. Applying the zero-knowledge transformation to a composed reduction would still only blind the initial instance-witness pair. However, applying the transformation independently to the first reduction and the second reduction and then composing would dictate that the prover blind both the initial instance-witness pair and the intermediate instance-witness pair, breaking the homomorphism requirement.

Nevertheless, we can show that certain transformations do satisfy the homomorphism property, meaning that, in a sense, they "respect" the composition operator for reductions of knowledge. In particular, we show that the *weak Fiat-Shamir transformation* can indeed be viewed as a functor. We additionally postulate the existence of a *succinct proof functor*, which asks whether a standard reduction of knowledge induces a corresponding reduction of knowledge where the witnesses are replaced with succinct proofs of their knowledge.

8.3.1 The Weak Fiat-Shamir Functor

Recall that the Fiat-Shamir transformation (Construction 7.1) takes a public-coin reduction of knowledge and produces a non-interactive reduction of knowledge in the random oracle model. Roughly, this is done by having both parties simulate the verifier's randomness privately by querying the random oracle on the prover's prior message. In the standard version, as presented in Section 7.1, the verifier additionally hashes the input instance to prevent a subtle class of attacks [62]. Unfortunately, this version does not satisfy the homomorphism property of a functor: Applying the Fiat-Shamir transformation after composing will only hash the initial instance pair. However, applying the transformation independently to both reductions first then composing would result in a reduction where the intermediate instance is also hashed.

To circumvent this, we can consider the Fiat-Shamir transformation as originally presented, where the input instance is not hashed [67]. We refer to this variant as the weak

Fiat-Shamir transformation and show below that it can be viewed as a functor. We note that we present the weak Fiat-Shamir transformation only for pedagogical purposes, and do not intend for this variant to be used for practical proof system design.

Construction 8.2 (Weak Fiat-Shamir Functor). Let ρ denote a random oracle. We construct the weak Fiat-Shamir functor F_{WFS} from the category of public-coin reductions of knowledge to the category of non-interactive reductions of knowledge in the random oracle model as follows. For arbitrary relation $\mathcal{R}$ we define $F(\mathcal{R}) = \mathcal{R}$. For arbitrary public-coin reduction $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ with ℓ rounds, we construct a non-interactive reduction of knowledge $F_{\mathsf{WFS}}(\Pi) = (\mathcal{G}, \mathcal{K}, \mathcal{P}', \mathcal{V}')$ in the random oracle model as follows.

$\underline{\mathcal{P}'(\mathsf{pk}, u_1, w_1) \to (u_2, w_2)}$:

1. Run $\mathcal{P}(\mathsf{pk}, u_1, w_1)$. On the ith message m_i, respond with verifier randomness $r_i = \rho(m_i)$. Let (u_2, w_2) be the output of $\mathcal{P}$ and let $\pi = (m_1, \ldots, m_\ell)$.
2. Send π to the verifier.
3. Output (u_2, w_2).

$\underline{\mathcal{V}'(\mathsf{vk}, u_1) \to u_2}$:

1. Receive $\pi = (m_1, \ldots, m_\ell)$ from the prover. Compute $r_i \leftarrow \rho(m_i)$.
2. Run $\mathcal{V}(\mathsf{vk}, u_1)$ with randomness $(r_1, \ldots, r_\ell)$. In round i send prover message m_i. Let u_2 be the output of $\mathcal{V}$.
3. Output u_2.

Lemma 8.2 (Weak Fiat-Shamir Functor). Construction 8.2 is a functor from the category of public-coin reductions of knowledge to the category of non-interactive reductions of knowledge in the random oracle model.

Proof. Given a public-coin reduction of knowledge $\Pi : \mathcal{R}_1 \to \mathcal{R}_2$ by Fiat and Shamir [67], we have that $F_{\mathsf{WFS}}(\Pi)$ is a non-interactive reduction of knowledge of type $\mathcal{R}_1 \to \mathcal{R}_2$ in the random oracle model.

We must argue the two required coherence conditions for F_{WFS}. Indeed, by observation for any two public-coin reductions $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2$ and $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3$ we have that

$$F_{\mathsf{WFS}}(\Pi_2) \circ F_{\mathsf{WFS}}(\Pi_1) = F_{\mathsf{WFS}}(\Pi_2 \circ \Pi_1).$$

Moreover, recall for any relation $\mathcal{R}$ the identity reduction $1_{\mathcal{R}} : \mathcal{R} \to \mathcal{R}$ simply outputs the prover and verifier's inputs. Then, we have that

$$F_{\mathsf{WFS}}(1_{\mathcal{R}}) = 1_{\mathcal{R}} = 1_{F_{\mathsf{WFS}}(\mathcal{R})}.$$

Therefore, we have that F_{WFS} is a functor. □

Assumption 8.1 (Random Oracle Instantiation Functor). Let the random oracle instantiation functor, F_{ROI}, be defined as $F_{\mathsf{ROI}}(\mathcal{R}) = \mathcal{R}$, and $F_{\mathsf{ROI}}(\Pi)$ be the result of instantiating the random oracle in Π with a cryptographic hash function. Then F_{ROI} is a functor from the category of reductions of knowledge in the random oracle model to the standard category of reductions of knowledge.

8.3.2 The Succinct Proof Functor

The categorical lens additionally enables us to ask questions which traditional cryptographic frameworks struggle to articulate. For instance, consider a succinct proof of knowledge Π for circuit-satisfiability (Definition 5.2). Then, for any NP relation $\mathcal{R}$, we can consider a corresponding relation, $\mathsf{SP}_{\Pi}(\mathcal{R})$, which checks succinct proofs for valid instances in $\mathcal{R}$. That is, we can consider a transformed relation where witnesses for statements are succinct proofs of the original (long) witness.

Definition 8.6 (Succinct Proof Relation). For a succinct proof of knowledge $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for circuit-satisfiability and for a binary NP relation $\mathcal{R}$, we define the corresponding succinct proof relation $\mathsf{SP}_{\Pi}(\mathcal{R})$ as

$$\mathsf{SP}_{\Pi}(\mathcal{R}) = \left\{ (\mathsf{pp}, u, \pi) \;\middle|\; \begin{array}{l} \mathsf{vk} \leftarrow \mathcal{K}(\mathsf{pp}, F_{\mathcal{R}}), \\ \mathcal{V}(\mathsf{vk}, u, \pi) = 1 \end{array} \right\}$$

where $F_{\mathcal{R}}$ is the predicate corresponding to $\mathcal{R}$.

Given such a relation, we ask the following question: Suppose a prover can interactively reduce to the task of checking $(u_2, w_2) \in \mathcal{R}_2$ given $(u_1, w_1) \in \mathcal{R}_1$. Then, does this induce a corresponding prover that can interactively reduce to the task of checking a *succinct proof* for u_2 given only a *succinct proof* for u_1? This question can be stated categorically as follows.

Conjecture 8.1 (A Succinct Proof Functor). There exists a succinct proof of knowledge Π for circuit satisfiability and a functor $F : \mathsf{RoK} \to \mathsf{RoK}$ such that $F(\mathcal{R}) = \mathsf{SP}_{\Pi}(\mathcal{R})$. That is, the following diagram commutes.

$$\begin{array}{ccc} \mathcal{R}_1 & \xrightarrow{\quad \Pi_{\mathsf{red}} \quad} & \mathcal{R}_2 \\ \Big\downarrow F & & \Big\downarrow F \\ \mathsf{SP}_{\Pi}(\mathcal{R}_1) & \xrightarrow{F(\Pi_{\mathsf{red}})} & \mathsf{SP}_{\Pi}(\mathcal{R}_2) \end{array}$$

While this conjecture may seem unremarkable at first glance, it would have sweeping implications for recursive proof systems (Chapter 5): It would allow us to build *inefficient* reductions where the IVC proof is the entire trace of the computation, and then generically derive *efficient* IVC schemes that match the space complexity of Valiant's original technique. If this conjecture is false, it would indicate that *all* efficient IVC schemes must rely in a non-blackbox way on the structural correspondence between input and output relations in each step.

8.4 The Yoneda Perspective

In 1960, mathematician Nobuo Yoneda, proved a simple, but subtly powerful result now known as the *Yoneda Lemma* [138]. In subsequent decades, the Yoneda Lemma arguably

became the most important result in category theory, with sweeping implications in all other branches of mathematics. At its essence, the Yoneda Lemma formally captures the following statement.

An object is equivalent to all of its projections.

In a sense, this statement captures the central dogma of category theory, which, unlike set theory, studies mathematical objects through their relation to other objects rather than through their contents. The Yoneda Lemma is so remarkably subtle that it can only be formally stated and proven in the categorical language. We have already seen how to define an "object" categorically, but how should we define "equivalent", and "projection" let alone "all projections"? We will see that the categorical language is powerful enough to formally capture each of these pieces.

Our goal for this section is to understand the Yoneda Lemma, and then understand what it implies for proofs of knowledge. At a high level, categorically, the projection of an object onto a second object is determined by the set of morphisms into the second object. As such, the Yoneda Lemma roughly states that an object is determined by all of its morphisms. This means that, in the setting of proofs of knowledge, Yoneda tells us that a relation is equivalent to all the ways it can be reduced into other relations. This, as we will demonstrate, will have several interesting consequences. One such consequence is that that a reductions of knowledge of type $\mathcal{R}_1 \to \mathcal{R}_2$ is equivalent to a linear transformation from a proof of knowledge for $\mathcal{R}_2$ to a proof of knowledge to $\mathcal{R}_1$.

To understand Yoneda, it helps to start with a concrete example that contains the germs of generality, and then generalize until the result becomes apparent. Indeed, let us fix a vector space $\mathbb{F}^n$ for finite field $\mathbb{F}$ and let $\mathbb{G}$ be an arbitrary group that is scaled by $\mathbb{F}$. We can naturally consider the dual vector space

$$(\mathbb{F}^n)^* \cong \mathbb{F}^n \to \mathbb{G}$$

which is the set of maps from $\mathbb{F}^n$ to $\mathbb{G}$ that are *polymorphic* (or *natural* in categorical terms) in $\mathbb{G}$. That is, the maps treat $\mathbb{G}$ as a black box and, consequentially, work over any instantiation of $\mathbb{G}$. Similarly, we can consider the double-dual space

$$(\mathbb{F}^n)^{**} \cong (\mathbb{F}^n \to \mathbb{G}) \to \mathbb{G}$$

which is the set of maps from the dual space $(\mathbb{F}^n)^*$ to $\mathbb{G}$ that are natural in $\mathbb{G}$.

Now, consider a map v^{**} in the double-dual space $(\mathbb{F}^n)^{**}$. This map cannot manifest an element in $\mathbb{G}$ on its own (e.g., by spitting out a hardcoded element), because it is required to be polymorphic over all possible instantiations of $\mathbb{G}$. Therefore, its only option is to use the input map f in the dual space $(\mathbb{F}^n)^* \cong \mathbb{F}^n \to \mathbb{G}$ to produce an element of $\mathbb{G}$. However, the map f requires an element v of $\mathbb{F}^n$ as input. But becase v^{**} takes no additional input, this means that v^{**} must manifest v on its own. That is, it must have v (or the information needed to produce it) hardcoded. As such, given any map v^{**} in $(\mathbb{F}^n)^{**}$ we can extract out the underlying element v in $\mathbb{F}^n$. Conversely, given any element v in $\mathbb{F}^n$ we can derive a corresponding map v^{**}. This correspondence is captured by the following isomorphism,

which formalizes a well-known result in linear-algebra that a vector space is equivalent to it's double-dual space.

$$\mathbb{F}^n \cong (\mathbb{F}^n \to \mathbb{G}) \to \mathbb{G}.$$

Generalizing, the above reasoning holds for any two objects A and B in a category $\mathbb{C}$. That is we have that A is equivalent to maps from morphisms in $\mathbf{Hom}(A, B)$ to B that are natural in B. Equationally, we have that

$$A \cong \mathbf{Hom}(\mathbb{C}(A, B), B)$$

where $\mathbb{C}(A, B)$ denotes the set of morphisms from A to B in $\mathbb{C}$. Now, we can vastly generalize the above equivalence with the following observation: for any set-valued functor $F : \mathbb{C} \to \mathsf{Set}$ (where Set is the category of sets and maps over these sets) given a morphism $m \in \mathbb{C}(A, B)$ and a wrapped element a' in the set FA, we can apply F to m to produce a new morphism $m' \in \mathsf{Set}(FA, FB)$, and apply m' to a' to produce a new wrapped element b' in FB. Then, by the prior reasoning, we have that the set FA is equivalent to the set of homomorphisms from $\mathbb{C}(A, B)$ to the set FB that are natural in B. That is, we have that

$$FA \cong \mathbf{Hom}(\mathbb{C}(A, B), FB).$$

We can denote naturality by replacing B with $\cdot$ as follows.

$$FA \cong \mathbf{Hom}(\mathbb{C}(A, \ \cdot\ ), F).$$

We can view $\mathbb{C}(A, \ \cdot\ )$ as a functor that maps an object B to $\mathbb{C}(A, B)$ and maps a morphism $f \in \mathbb{C}(B, C)$, to a morphism Ff, which, given a morphism $g \in \mathbb{C}(A, B)$, produces the morphism $f \circ g \in \mathbb{C}(A, C)$. As such, the above homomorphism can be viewed as a natural transformation. The functor $\mathbb{C}(A, \ \cdot\ )$ is often referred to as the Yoneda embedding and is denoted as $Y(A)$. The functor $Y(A)$ can be viewed as containing all the projections of A onto all other objects B, and thus can be viewed as containing all of the information of A, at least from the perspective of category $\mathbb{C}$. This gives us the famous Yoneda Lemma.

Lemma 8.3 (Yoneda [138]). For category $\mathbb{C}$ and functor $F : \mathbb{C} \to \mathsf{Set}$, we have that

$$FA \cong \mathbf{Hom}(Y(A), F).$$

By applying the Yoneda Lemma to the category of reductions of knowledge, we immediately get the following corollary, which essentially states that a relation $\mathcal{R}_1$ is equivalent to homomorphisms from reductions of type $\mathcal{R}_1 \to \mathcal{R}_2$ to $\mathcal{R}_2$ that are polymorphic in $\mathcal{R}_2$.

Corollary 8.1 (Yoneda, RoK). For any relation $\mathcal{R}$ in RoK, and functor $F : \mathsf{RoK} \to \mathsf{Set}$ we have that

$$F\mathcal{R} \cong \mathbf{Hom}(Y(\mathcal{R}), F).$$

Setting the functor F to be $Y(\mathcal{R}_1)$ we immediately get the following theorem. At a high level, it states that reductions of knowledge of type $\mathcal{R}_1 \to \mathcal{R}_2$ can be understood as homomorphisms from reductions of type $\mathcal{R}_2 \to \mathcal{R}_3$ to reductions of type $\mathcal{R}_1 \to \mathcal{R}_3$ that are polymorphic in $\mathcal{R}_3$. For example, by setting $\mathcal{R}_3 = \mathcal{R}_\top$, we see that a reduction of knowledge of type $\mathcal{R}_1 \to \mathcal{R}_2$ is a linear transformation from a proof of knowledge for $\mathcal{R}_2$ to a proof of knowledge for $\mathcal{R}_1$.

Theorem 8.2 (Proofs of Knowledge are Maps over Propositions of Knowledge).

$$\mathsf{RoK}(\mathcal{R}_1, \mathcal{R}_2) \cong \mathbf{Hom}(Y(\mathcal{R}_2), Y(\mathcal{R}_1)).$$

Theorem 8.2 formally captures the central insight of this thesis. Traditionally, cryptographers would design complex proofs of knowledge for, say, R1CS, or polynomial evaluations, or inner-products, by transforming (or compiling) simpler proofs of knowledge. The most natural example of this is the proof of knowledge for inner-products due to Bootle et al. [36], which internally invokes a proof of knowledge for inner-products for half-sized vectors. As such, this can be viewed as transforming a proof of knowledge for a size $n/2$ inner-product into a proof of knowledge for a size n inner-product. This thesis petitions to treat these transformations themselves as the central object of study, rather than the proof systems that they work over. Mathematically, this can be viewed as stepping from the right side of the equation in Theorem 8.2 to the left side. As with Yoneda's lemma, this has served as a simple, but, as we have hopefully demonstrated, subtly powerful new perspective.

Chapter 9

Prospects

This chapter contains ideas from discussions with Sanjam Garg, Bryan Parno, Leah Rosenbloom, Srinath Setty, Justin Thaler, and Psi Vesely.

The art of doing mathematics consists in finding that special case which contains all the germs of generality.

– David Hilbert (apocryphally)

In this thesis, we have provided a theory of composition for proofs of knowledge and thereby have provided a mathematical account of the proofs-as-maps paradigm. Using this theory as a succinct framework, we have designed and analyzed a variety of novel proof systems with an emphasis on recursive proof systems. Today, we are seeing that recursion has become a cornerstone technique in developing modern proofs of knowledge, and that the proofs-as-maps perspective formalized by our theory is becoming a quickly adopted design paradigm for achieving this end. In this chapter, we discuss the prospects of both our techniques and the underlying theory for interactive proof theory and cryptography in general.

9.1 A Plan for zkSNARKs for Universal Machines

As discussed in Section 1.1, modern interest has centered around a particularly compelling application: proving the correct execution of a *universal machine* [6, 7, 8, 9]. This enables a single-proof system for the underlying substrate running all applications. The way zkSNARKs are developing today is much like how CPUs developed half a century ago: Just like CPU hardware in the 1950s, in the early 2010s we designed optimized SNARKs for domain specific applications such as verifiable certificates, private blockchains, or verifiable database. But in the late 2010s, we started building SNARKs for general purpose universal machines much in the same the same way CPUs evolved towards general purpose computing. And the key benefit at this stage of evolution is scalability at the expense of some latency due to the additional abstraction overhead. And today, limited by raw computational capabilities, we are beginning to develop optimized SNARKs for the particular application of universal machines, much like how transistor-limited CPUs today must leverage new microarchitectural techniques (such as pipelining, caching, and specialized co-processors) to actually meet modern demands.

Recall that Valiant [133] proposed incrementally verifiable computation (IVC), which reflected the recursive structure of the computation into the proof itself: Given a short proof π_i attesting to i steps of computation, the prover can write a short proof π_{i+1} that attests to $i+1$ steps by proving the correct execution of an arithmetic circuit that runs the latest step of computation, and checks π_i (using the proof system's verifier). This avoids having to fix the recursion depth ahead of time, while ensuring that the prover's memory overhead only scales with a single step of execution.

Ben-Sasson et al. [27], following Valiant's idea of recursively embedding proofs [133], demonstrated that the problem of proving universal machine execution reduces to the more fundamental problem of proving recursive applications of a function F. In particular, a prover could write a succinct proof about $i+1$ steps of a CPU proving the latest execution of the CPU as well as proving that there exists a proof for i steps of the CPU. This avoids having to fix the recursion depth ahead of time, while ensuring that the prover's memory overhead only scales with a single step of execution. Since the initial approach of Ben-Sasson et al., research effort has largely focused on reducing the overhead of recursion (Section 1.1.6).

The technical developments in this thesis can be viewed as making important progress in this program. In the Nova proof system (Section 4.2, Section 5.1), following the ideas of Bowe et al. [38], we showed that it is possible to use reductions of knowledge, namely folding schemes, as opposed to proofs of knowledge to recursively compress statements (and witnesses) of correct execution for each step into a running statement and witness (which do not grow in size) that can be checked at the end. This allows us to avoid expensive proof machinery altogether. In Section 5.1.5, we empirically demonstrated that folding is substantially more efficient than prior approaches [27, 35, 38, 45], bringing recursive proofs significantly closer to practice.

In Section 5.2, we developed non-uniform IVC, which enables a more efficient folding-based recursive proof system for universal machines. In particular, to prove universal machine execution using IVC, the recursive function F can encode all possible "instructions"

that can be run in each step of execution (e.g., add, mul, load, store). Ideally, however, the prover runtime should scale with the particular instruction run in each step, rather than with the full set of instructions. This issue is solved by non-uniform IVC, which modifies our original recursive proof system to prove a different function in each step, while maintaining that these functions are scheduled based on the program logic. In Section 4.3 we developed a folding scheme for high-degree constraints, and thereby using the techniques of Section 5.1 and 5.2 this enables recursive proof systems for universal machines represented using high-degree constraint systems. A key challenge is that the folding verifier circuit must perform elliptic curve operations, which are expensive to represent as field operations in the circuit. To address this, Kothapalli and Setty [98] demonstrate that we can utilize a cycle of elliptic curves (where the elliptic curve operations of one curve can be efficiently represented as field operations on the other and visa-versa) to outsource the expensive elliptic curve operations to the secondary curve and instead efficiently fold a proof of correct execution on the primary curve.

As we work to scale the first generation of folding-based recursive proof systems to real computational workloads, we are presented with various growing pains. Our research objective is to develop design paradigms that attend to the challenges faced by industrial applications. This objective can be met by **(1)** reducing the recursion overhead, **(2)** designing techniques to more efficiently encode computation, and **(3)** developing better cryptographic models to reason about recursion and the underlying assumptions (as initiated in Section 7.2).

9.1.1 Reducing the Recursion Overhead

A growing body of recent works [32, 41, 65, 145] further optimize the above techniques in service of reducing the *recursion overhead*, that is, the size of the verifier circuit in each step of recursion. Today, HyperNova (the IVC scheme that results from the folding scheme in Section 4.3), affords one of the most attractive efficiency tradeoffs with only a linear dependence on the degree of constraints and instances being folded. However, this is in exchange for a logarithmic number of hash operations, which may be expensive in practice. We propose to further improve this to just a constant number of hashes by re-purposing the underlying *sumcheck protocol* [106] as a mechanism to *fold* two sets of constraints rather than as a mechanism to *linearize* a set of constraints as in HyperNova.

Moreover, all prior techniques do not address additional costs to achieve *zero-knowledge IVC* (i.e., the IVC proof reveals no information about the underlying witness), which can *double* the recursion overhead [45]. In particular, existing techniques must either randomize the witness for *each* step of execution, or write a zero-knowledge proof of a valid IVC proof at the end of execution. Both of these techniques are concretely expensive [45, 101]. To remedy this, we propose to blind the IVC proof in just the *final* step of recursion using a blinding folding scheme, significantly cutting the overhead of zero-knowledge. This blinded IVC proof can then be further incremented by another untrusted prover, enabling a larger class of distributed applications.

9.1.2 More Efficiently Encoding Computation

Improvements to the recursion overhead only addresses a piece of designing efficient zero-knowledge universal machines. Modern effort is also focused on designing more efficient encodings of computations and designing recursive proof systems that target these encodings [14, 73, 114, 127]. As an example, HyperNova designs a folding scheme for high-degree constraints enabling computations to be expressed in fewer constraints. We propose several improvements in this vein.

More Efficiently Supporting Multiple Instructions We propose to improve our approach of non-uniform IVC for supporting multiple instructions. In particular, one caveat with existing folding schemes [41, 99, 101] is that they can only fold statements regarding a single function F. This is reflected in our approach which allows for different functions by maintaining a separate running statement for each function. Unfortunately, this is still concretely expensive. Instead, we propose to fold two statements that refer to different functions by compressing the corresponding constraints much in the same way we compress inputs to these constraint systems. This enables a single running statement for all invoked functions.

Smaller Circuits to Encode Instructions Even with efficient support for multiple instructions, many instructions are still too expensive to directly encode as an arithmetic circuit. To address this, we propose to incorporate Jolt [14], which show that it is often more efficient to implement circuits that query a hard-coded evaluation table rather than evaluate the instruction on the fly. As such, Jolt can be viewed a front-end for encoding universal machine computations, which can then be proven using the recursive proof systems presented in this work. In particular, in each cycle instead of directly invoking an instruction, Jolt invokes the Lasso lookup proof [126]. To handle reads and writes to RAM (and to registers) Jolt uses a memory-checking proof from Spice [134]. Both the lookup proof and the memory-checking logic in each step can be encoded in a minimal R1CS instance (roughly 60 constraints per cycle). Jolt considers the full unrolled circuit of the universal machine, and proves this data-parallel circuit using a variant of the SIMD Spartan proof system presented in Section 6.2.

While Jolt's current backend proof system enables a remarkably efficient proof system for smaller computations, the prover's memory overhead will scale with the entire trace of the virtual machine, making it challenging to scale for more complex programs. Here, we propose to use Nova as a backend proof system instead, enabling a prover to incrementally prove each cycle of the Jolt, which in turn ensures that its memory overhead scales with a single cycle. Of course, the recursion overhead of Nova is roughly 10000 gates (Section 5.1.5), which dwarfs the number of constraints to represent a single cycle of Jolt. This can be amortized away by utilizing Nova to prove a large (but manageable) batch of cycles in each recursive step (e.g., 1000 cycles).

One limitation is that the Jolt frontend only supports instructions that are *decomposable*, that is, can be represented as an efficient function over the results of much smaller lookups. One of the major contributions of Jolt is to demonstrate that the entire RISC-V

instruction set satisfies this requirement. However, this is not the case for the Ethereum virtual machine instruction set, which supports undecomposable instructions such as a Keccak-256 hash. To account for this, we can utilize non-uniform IVC, to selectively employ Jolts frontend only for the instructions which are decomposable.

More Efficiently Supporting Conditionals For some applications instructions can be larger course-grained computations (e.g., SHA256). In such a setting, instructions may include conditionals that may not activate the full circuit. Here, we propose to use the dynamic commitments approach of Sublonk [57] to enable the prover to prove only activated portions of each instruction in each step of recursion. In more detail, Sublonk demonstrates how to dynamically compute commitments to the active part of a circuit and efficiently prove that this commitment is well-formed using lookup proofs [72]. Given these dynamic commitment, the correct execution of the activated sub-circuit can then be proven using a standard proof system such as Plonk [73] or folded.

9.1.3 Polynomial-Depth Recursion

While recursive proofs enable a broad class of modern distributed applications, a crucial issue is that for all existing approaches to IVC (in standard models) the soundness error grows exponentially with the recursion-depth, even in the presence of knowledge assumptions (following the reasoning in Section 5.1.1). As applications today are using IVC with no strict bounds on depth the question of polynomial-depth IVC is a pressing matter both in theory and in practice.

Moreover, as far as we know, recursion requires proving statements about the proof system's verifier, which is implausible if the verifier queries a random oracle [87]. Thus, regardless of recursion-depth, Valiant first assumes that there exists a succinct non-interactive argument of knowledge (SNARK) in the plain model (which are only provably known in the random oracle model). Then, Valiant additionally assumes that this SNARK's extractor only has a constant multiplicative overhead over the prover to achieve logarithmic-depth recursion. To justify this assumption, Valiant demonstrates that there exists a straight-line SNARK in the random oracle model. Valiant then, justifiably, assumes that this SNARK can be instantiated with a cryptographic hash function to derive a SNARK in the plain model with constant multiplicative overhead in the extractor.

Modern folding-based IVC techniques must make more aggressive assumptions. As with SNARKs, we only know of folding schemes in the random-oracle model [35, 45, 101]. Thus, as with Valiant, such schemes must be heuristically instantiated in the plain model. Similarly, to achieve logarithmic-depth recursion, folding-based IVC schemes must assume the existence of folding schemes with constant multiplicative overhead in the extractor. Unlike Valiant, however, these works do not provide a corresponding straight-line folding scheme in the random oracle model as justification. Thus, folding-based IVC schemes only realistically achieve *constant-depth* recursion.

In a feasibility result, Choudhuri, Jain, and Jin [56] (following Kalai, Paneth, and Yang [91]) show that such heuristic assumptions can certainly be avoided by settling for IVC limited to *deterministic* computations (thereby avoiding the need for an extractor). Even

in this setting, the best known construction [64] achieves proof sizes that grow additively with the recursion-depth, meaning that succinctness is only satisfied for logarithmic-depth recursion.

Valiant shows that we can still recursively prove polynomial-time computations by rearranging steps of computation along nodes of a binary tree. Essentially, instead of having a proof of n steps attest to a a proof of $n-1$ steps and the latest execution of F, it instead attests to two consistent proofs of $n/2$ steps (thereby maintaining logarithmic-depth recursion). Bitansky et al. [30] formalize this technique as a general compiler that takes any polynomial-time computation and encodes it as a statement provable with logarithmic-depth recursion under the same assumptions as Valiant. While this approach pragmatically enables proofs for arbitrary polynomial-time computations, it sidesteps the fundamental question of the soundness of polynomial-depth recursion.

Indeed, Hall-Andersen and Nielsen [87] indicate that polynomial-depth recursion may be infeasible in the random-oracle model without additional knowledge assumptions. In light of this, Chen et al. [52] introduce the *arithmetic random oracle model* (AROM) where the random oracle is additionally equipped with a predicate oracle checking random oracle query-response pairs and an algebraic extension (that operates over field elements instead of bitstrings) of this predicate oracle. Chen et al. show that we can achieve *provably secure* polynomial-depth IVC in this model. An attractive feature of this approach is that it avoids heuristic knowledge assumptions altogether. As a nascent — but certainly promising — model, the AROM invites further study to fully understand the precise security guarantees and how it may be effectively instantiated.

We propose to instead achieve polynomial-depth recursion by building straight-line IVC by utilizing a straight-line folding scheme, which ensures that the composed extractor only has an *additive* overhead with each recursive step. Using the straight-line transformation in Section 7.2, we can convert any of the proposed folding schemes into a straight-line folding scheme. A central challenge however, is that Fischlin's transformation seems inherently attached to the random oracle model, which, as Hall-Andersen et al. suggest, is incompatible with recursive proof systems. Translating the straight-line property and the corresponding transformation to a setting without the random oracle will be the bulk of our proposed technical contribution.

9.2 A Plan for Interactive Proof Theory

A central perspective championed by this thesis is a composition-first (i.e., categorical) approach for proofs of knowledge, where complex proof systems are designed by composing simpler proof systems. As proof systems grow in complexity, our theory of composition presented is quickly being adopted in the literature [33, 114, 115, 121, 145, 146].

Outside of managing complexity, a theory of composition allows us to expediently make sweeping observations about all proofs of knowledge. As a motivating example, we utilized Yoneda's perspective [138] to demonstrate that reductions of knowledge from relation $\mathcal{R}_1$ to relation $\mathcal{R}_2$ are equivalent to (linear) transformations from a proof system for $\mathcal{R}_2$ to a proof system for $\mathcal{R}_1$. This allows us translate from one perspective to another whenever

convenient. While observations in this style may stand on their own, our intention is that they enable faster development for concrete techniques. Below, we identify two directions for extending our theory of composition.

9.2.1 Generalizing Existing Results

Just as we have generalized proofs of knowledge to reductions of knowledge, we can generalize various auxiliary results, such as transformations, idealized models, and impossibility results over proofs of knowledge to the setting of reductions of knowledge. As discussed in Section 1.3, we can expect reductions of knowledge to be compatible with sufficiently generalized idealized soundness models such as the random oracle model and the algebraic group model, idealized communication models such as interactive oracle proofs, and heuristic transformations.

Indeed, we have already generalized the Fiat-Shamir transformation to reductions of knowledge (Section 7.1), and demonstrated that a variant known as the weak Fiat-Shamir transformation (Section 8.3) satisfies an additional homomorphism property, where transforming then composing is the same as composing then transforming. In Chapter 8, we formalized this property as functorality. We aim to generalize various other transformations to the setting of reductions, and demonstrate that they satisfy the functorality property.

Interactive oracle proofs (IOPs), as introduced in Section 1.1, are a particularly useful idealized model to generalize. Recall that IOPs consider the setting where a prover is allowed to send oracles as messages to the verifier, which in turn is allowed to query these oracles at any point. Ben-Sasson et al. [26], demonstrate that any proof of knowledge in the IOP model induces a corresponding non-interactive proof of knowledge in the random oracle model by instantiating the oracles with polynomial commitments. IOPs enable the core information theoretic portion of a proof of knowledge to be considered separately from the cryptographic machinery (i.e, polynomial commitments). As such, we would ideally use the reductions of knowledge framework in conjunction with IOPs to further simplify the conceptual burden for developing new proof systems. For this to be sound however, we must verify **(1)** that reductions of knowledge can indeed be expressed in the IOP model, **(2)** that two reductions in the IOP model can be composed to produce a new reduction in the IOP model, and for convenience **(3)** that the transformation of Ben-Sasson et al. satisfies functorality. We can similarly generalize various other models such as the generic group model, algebraic group model, and polynomial IOPs.

9.2.2 More Expressive Notions of Composition

Using our categorical developments as an ambient framework, we can begin to construct more expressive notions of composition, where the composition operation itself encodes non-trivial constraints or computations. This allows us to more faithfully capture the behavior of the underlying reductions and construct more complex proof systems as a result.

We have already seen in Section 2.5 how to augment our basic theory of composition with *refinement types* [70], which augments reductions of knowledge with customizable

preconditions on the input instances and postconditions on the output instances. We use this additional expressivity to capture more complex notions in the reductions of knowledge framework such as incrementally verifiable computation. If we take refinement types to their logical extreme, we arrive at fully *dependent types* [109], where the output relation type of a reduction can *depend* on the reduction itself.

In more detail, for infix binary relation $\sim$ recall that a refined reductions of knowledge $\Pi : \mathcal{R}_1 \xrightarrow{\sim} \mathcal{R}_2$ guarantees that for every input statement u_1 to Π, we will have that the reduction will output a statement u_2 such that $u_1 \sim u_2$. The binary relation $\sim$ can be viewed as constraining the set of permissible output statements in the relation $\mathcal{R}_2$. Dependent types more generally allow us to pick a new relation entirely based on the input statement (and witness). In particular, suppose we have a relation $\mathcal{R}_2$ that is *characterized* by instance-witness pairs (u_1, w_1) in $\mathcal{R}_1$ (denoted $\mathcal{R}_2(u_1, w_1)$). Then we can write

$$\Pi_1 : ((u_1, w_1) \in \mathcal{R}_1) \to \mathcal{R}_2(u_1, w_1)$$

to mean that on input (u_1, w_1) the reduction Π_1 outputs a new instance-witness pair (u_2, w_2) in a new relation $\mathcal{R}_2$ characterized by (u_1, w_1). We can refer to such reductions as *dependent reductions of knowledge*, and we can see immediately they generalize refined reductions of knowledge by considering the special case where

$$\mathcal{R}_2(u_1, w_1) = \{(u_2, w_2) \in \mathcal{R}_2 \mid u_1 \sim u_2\}.$$

Given *deterministic* dependent reductions of knowledge

$$\Pi_1 : ((u_1, w_1) \in \mathcal{R}_1) \to \mathcal{R}_2(u_1, w_1),$$
$$\Pi_2 : ((u_2, w_2) \in \mathcal{R}_2(u_1, w_1)) \to \mathcal{R}_3(u_2, w_2),$$

we have that

$$\Pi_2 \circ \Pi_1 : ((u_1, w_1) \in \mathcal{R}_1) \to \mathcal{R}_3(\Pi_1(u_1, w_1)).$$

A key observation here is that the output relation type of $\Pi_2 \circ \Pi_1$ depends on the result of executing Π_1. This marks the true power of dependent type systems, which enable the type system itself to perform computations, blurring the distinction between specification and implementation. Of course, this additional power comes at the expense of additional complexity. One of the key benefits of our basic theory of composition is that is abstracts away unnecessary details of the reduction in the type specification. If not used judiciously, dependent types risk re-exposing all of these details due to their sheer expressivity.

By the discussions in Section 8.4, we have that a reductions of knowledge of type $\mathcal{R}_1 \to \mathcal{R}_2$ represents a constrained set of transformations over proofs of knowledge, namely the set of *linear* transformations from a proof of knowledge for $\mathcal{R}_2$ to a proof of knowledge for $\mathcal{R}_1$. We can vastly generalize our theory of linear transformations over proofs of knowledge, by developing the theory of *non-linear* transformations over arbitrary reductions of knowledge. Namely, we can consider general transformations that construct a reduction of knowledge of type $\mathcal{R}_1 \to \mathcal{R}_4$ by internally invoking a reduction of knowledge of type $\mathcal{R}_2 \to \mathcal{R}_3$ as many times as needed. The composition operator in this setting, would be subroutine-substitution. Once again, while such a theory is more expressive, it will also be more difficult to work with and extend.

Yet another way we can extend the basic theory of composition is by considering *monadic composition*. Monadic composition can be understood as "skew" composition, where the output type of the first reduction does not perfectly match the input type of the second reduction, but this discrepancy is nevertheless handled by the composition operator itself, which perform the necessary intermediate processing. For instance, we already implicitly utilize the *option monad* in our standard composition operator, because the verifier can either output a reduced instance or abort (i.e., output $\perp$). Implicitly then, we have that $\Pi_2 \circ \Pi_1$ denotes the protocol where the prover and verifier first run Π_1, and abort if the output is $\perp$ or run Π_2 otherwise. As a more interesting example, consider reductions $\Pi_1 : \mathcal{R}_1 \to \mathcal{R}_2^\mu$ and $\Pi_2 : \mathcal{R}_2 \to \mathcal{R}_3^\nu$ for arbitrary $\mu, \nu \in \mathbb{N}$. We cannot directly compose Π_1 with Π_2 because Π_1 outputs a list of elements in $\mathcal{R}_2$ to be checked, whereas Π_2 only reduces a single element in $\mathcal{R}_2$. This can be naturally solved by defining a new composition operator $\diamond$ and define $\Pi_2 \diamond \Pi_1$ to mean mean that on input $(u_1, w_1) \in \mathcal{R}_1$, the prover and verifier first run Π_1 to produce a list of instance-witness pairs to be checked in $\mathcal{R}_2^\mu$, and then run μ parallel instances of Π_2 to reduce each of these pairs into a new list of instance-witness pairs to be checked in $\mathcal{R}_3^\nu$. Concatenating all of these sublists, we have that type of $\Pi_2 \diamond \Pi_1$ is $\mathcal{R}_1 \to \mathcal{R}_3^{\mu \cdot \nu}$. This particular composition operator is called the *list monad*, which is just one of many potential monads we can devise to succinctly express complex reductions.

9.3 A Plan for Cryptography

In this section, we propose a research program for cryptography as a whole. Just as we have lifted proofs of knowledge into a categorical framing, we propose that, step-by-step, we can do the same for a much broader swath of modern cryptography. Speculative as it surely is, if successful, this would afford a general theory of composition for cryptography that is informed by well-trodden paradigms in type theory, category theory, and abstract algebra.

Starting back in the 1980s, luminaries such as Blum [31], Micali, and Goldwasser [82, 83] set the stage for modern cryptography: Starting with the definition of semantic security [82], we began to treat cryptographic systems as secure so long as they were secure against a *polynomial-time* adversary with overwhelming probability. This relaxation is central to modern cryptography, in which most schemes are based on the hardness of a problem for a polynomial-time adversary. As we expanded the theory of cryptography, we began to study the security of cryptographic protocols under various adversarial classes with varying time and space complexity, thus cementing modern cryptography in the language of computational complexity theory.

While complexity theory has been a wildly successful for developing the theory of cryptography, it is now beginning to strain and buckle under the pressure of modern large scale cryptographic systems. Complexity theory, being a low-level language, working primarily over Turing machines which map bitstrings to bitstrings, with overly broad specifications (e.g., polynomial time, exponential space, logarithmic communication) does not provide a powerful enough specification language to reason about modern cryptographic protocols

modularly. The lack of a specification language for Turing machines means that when composing Turing machines, we must often study the entire system globally to understand their behavior. This lead to an an era in the 1980s and 1990s where we were hand-crafting specifications (such as semantic security) in the low-level language of complexity theory for various cryptographic primitives, and then manually reasoning how primitives satisfying these specifications can be composed.

This state of affairs was tackled by Canetti's *universal composability* framework [50] in the early 2000s, which defines the specification of a cryptographic protocol as a trusted third party, called a *functionality*, that captures the idealized behavior of the protocol. As a functionality can implement virtually any behavior, virtually any cryptographic primitive can be defined and analyzed in the universal composability framework. Canetti then defines subroutine substitution as a secure composition operation, where calls to a functionality $\mathcal{F}$ in a larger protocol are replaced with calls to a subprotocol which realizes $\mathcal{F}$.

Mauer's *constructive cryptography* [112], though not explicitly stated, similarly achieves a general theory of composition for cryptography by taking a much more categorical approach: cryptographic systems are treated as transformations from a weaker cryptographic resource (such as a public communication channel) to a stronger cryptographic resource (such as a private communication channel). Broadbent and Karvonen [40], more explicitly cast this framework categorically by treating resources as objects and protocols as maps.

Canetti's theory of composition has made remarkable progress in standardizing cryptography, and Mauer, Broadbent, and Karvonen, make important early progress into lifting cryptography from the low-level language of complexity theory to the higher-level language of category theory. However, these theories are limited from two central drawbacks: First, the specification language is not formalized (i.e., does not have a formal syntax and semantics) in any of these theories. As a result cryptographers using these theories currently specify the behavior of these resources or functionalities in various ad-hoc pseudocode languages that vary in rigor. This issue is not fundamental however, and there are already research programs underway for formalizing the specification language for such theories [18, 51, 74, 105].

The second, and more fundamental limitation, is that all such general theories of composition are *overly general*. As there is no clear distinction between a specification a cryptographic specification, all such frameworks, by definition, must be expressive enough to specify arbitrary systems, making them no more effective in the specialized cryptographic setting. In particular, these framework are so general that it is difficult to make sweeping statements about *all* protocols formalized in this framework, which would be equivalent to making a sweeping statement about computation in general.

This is in sharp contrast with the reductions of knowledge framework, which study a sufficiently constrained subdomain of cryptography, namely proofs of knowledge. As a result, we can provide a formal security specification, and more importantly, we can develop a non-trivial metatheory that applies to all the protocols that fit into our framework. For instance, in Chapter 7 we develop transformations over any protocol (with mild qualifications) specified in our framework. In Section 8.4, we observe that protocols specified in our framework can be viewed as *linear transformations*. In Section 9.2 above, we explore the possibility of more expressive composition operations, which handle computational

artifacts specific to proofs of knowledge. Moreover, by lifting into the categorical realm in Chapter 8, we can very cleanly label these meta-theoretic results as functors, natural transformations, monads, and so on.

Our goal is to slowly and thoughtfully generalize this categorical approach to increasingly broader areas of cryptography. Naturally, the meta-theory will become increasingly sparse as we broaden our scope, however this approach will give us a much finer knob to tune the expressivity to just as much as is needed for any particular domain of cryptography. As a starting point, multi-party computation (MPC) in particular can be viewed as a generalization of proofs of knowledge to multiple parties that jointly compute a single function while hiding their private inputs. As such we can expect to define a *super-category* of MPC schemes. Given such a category, we can start to develop a (more limited) metatheory for MPC schemes using the algebraic frameworks already presented in category theory, which would immediately apply to proofs of knowledge.

We began this thesis with the observation that the unifying theme of modern cryptography is an emphasis on the integrity and privacy of *computation* as opposed to just *data*. We can hope that this unifying theme can help uncover a coherent hierarchical relationship for disparate primitives such as fully homomorphic encryption, oblivious RAM, proofs of knowledge multi-party computation, and indistinguishability obfuscation. As a result, we could hope to discover a unifying categorical framework for a large swath of modern cryptography. We will then be free to import tools and perspectives from category theory, type theory, and constructive logic to inform cryptography as a whole. This would lift cryptography into the realm of higher algebra, completing a transition that has become inevitable today for nearly every domain of mathematics.

Bibliography

[1] Bellperson. https://crates.io/crates/bellperson. 5.1.5

[2] Neptune. https://crates.io/crates/neptune. 5.1.5

[3] Nova: Recursive SNARKs without trusted setup. https://github.com/Microsoft/Nova. 5.1.5

[4] Pasta curves. https://crates.io/crates/pasta_curves, . 5.1.5

[5] Pasta-MSM. https://crates.io/crates/pasta-msm, . 5.1.5

[6] Polygon zkEVM. https://polygon.technology/polygon-zkevm. 1.1.6, 9.1

[7] RISC Zero. https://www.risczero.com. 1.1.6, 5.2, 9.1

[8] Scroll zkEVM. https://scroll.io. 1.1.6, 9.1

[9] zkSync zkEVM. https://zksync.io. 1.1.6, 9.1

[10] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1):1–54, 2009. 1.1.2

[11] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In *Annual Cryptology Conference*, pages 209–236. Springer, 2010. 3.1.3, 3.3.2, 3.3.3

[12] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009. 1.3

[13] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998. 1.1.3

[14] Arasu Arun, Srinath Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. *Cryptology ePrint Archive*, 2023. 1.1, 9.1.2, 9.1.2

[15] Giuseppe Ateniese, Michael T Goodrich, Vassilios Lekakis, Charalampos Papamanthou, Evripidis Paraskevas, and Roberto Tamassia. Accountable storage. In *Applied Cryptography and Network Security: 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings 15*, pages 623–644. Springer, 2017. 1.1

[16] Thomas Attema and Ronald Cramer. Compressed-protocol theory and practical application to plug & play secure algorithmics. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III*, pages 513–543. Springer, 2020. 1.1.5, 1.2, 1.3.1, 1.4, 1.4, 4

[17] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. Compressed Sigma-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV*, pages 526–556. Springer, 2021. 1.4, 1.4, 4

[18] Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Adrien Koutsos, and Pierre-Yves Strub. Mechanized proofs of adversarial complexity and application to universal composability. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2541–2563, 2021. 9.3

[19] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 263–280. Springer, 2012. 1.1.5, 4

[20] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *Annual International Cryptology Conference*, pages 390–420. Springer, 1992. 1.1.1

[21] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993. 1.1.3, 1.3

[22] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In *Advances in Cryptology—CRYPTO'88: Proceedings 8*, pages 37–56. Springer, 1990. 7.3

[23] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 401–414, 2013. 5.2

[24] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Annual cryptology conference*, pages 90–108. Springer, 2013. 1.1.5, 5.2

[25] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct {Non-Interactive} zero knowledge for a von neumann architecture. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 781–796, 2014. 1.1.6, 1.5, 5.2

[26] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part II 14*, pages 31–60. Springer, 2016. 1.1.3, 1.3, 9.2.1

[27] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79(4):1102–1160, 2017. 1.1, 1.1.6, 1.5, 1.5, **??**, 5.1.5, 5.2, 9.1

[28] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. 1.1, 4.3

[29] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd innovations in theoretical computer science conference*, pages 326–349, 2012. 1.1.5, 1.3.1

[30] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 111–120, 2013. 1.1.6, 1.2, 2.2, 1, 5.1.5, 9.1.3

[31] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983. 9.3

[32] Dan Boneh and Binyi Chen. Latticefold: A lattice-based folding scheme and its applications to succinct proof systems. *Cryptology ePrint Archive*, 2024. 9.1.1

[33] Dan Boneh and Binyi Chen. Latticefold: A lattice-based folding scheme and its applications to succinct proof systems. *Cryptology ePrint Archive*, 2024. 1.1.6, 4, 9.2

[34] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018. 1.5

[35] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*, pages 649–680. Springer, 2021. 1.1.4, 1.2, 1.4, 1.5, 1.5, 4, 5.1.5, 7.3, 7.1, 7.3, 9.1, 9.1.3

[36] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 327–357. Springer, 2016. 1.2, 1.3.1, 1.3.3, 1.3.3, 1.4, 1.4, 1.4.1, 2.3, 2.3, 4, 7.1, 8.4

[37] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. Sumcheck arguments and their applications. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*, pages 742–773. Springer, 2021. 1.2, 1.3.3, 1.2, 1.4, 1.4, 2.3, 2.2, 3.5, 3.1.3, 3.2.2, 3.5, 3.7, 3.10

[38] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. *IACR Cryptol. ePrint Arch.*, 2019. 1.1.6, 1.2, 1.5, 1.5, 1.5.2, 4, 5.1.5, **??**, 5.1.5, 9.1

[39] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964. IEEE, 2020. 1.5

[40] Anne Broadbent and Martti Karvonen. Categorical composable cryptography. In *International Conference on Foundations of Software Science and Computation Structures*, pages 161–183. Springer International Publishing Cham, 2022. 9.3

[41] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. *Cryptology ePrint Archive*, 2023. 1.1.6, 4, 9.1.1, 9.1.2

[42] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018. 1.1.5, 1.2, 1.3.3, 1.4, 2.3, 4, 1, 7.3.2

[43] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. *Cryptology ePrint Archive*, 2020. 1.5, 5.1.5, **??**, 5.1.5

[44] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 677–706. Springer, 2020. 1.1.3, 1.3, 1.4, 1.5

[45] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*, pages 681–710. Springer, 2021. 1.1.6, 1.2, 1.3.1, 1.3.3, 1.5, 1.5, 1.5.2, 2.2, 2.3, 4, 5.1.5, 7.3, 9.1, 9.1.1, 9.1.3

[46] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology*

and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27, pages 65–97. Springer, 2021. 1.1.5, 1.2, 1.4, 1.4, 4, 2

[47] Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, and Luca Nizzardo. Vector commitment techniques and applications to verifiable decentralized storage. *IACR Cryptol. ePrint Arch.*, 2020:149, 2020. 1.5

[48] 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 *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33. Springer, 2021. 1.2, 1.3

[49] Matteo Campanelli, Anca Nitulescu, Carla Ràfols, Alexandros Zacharakis, and Arantxa Zapico. Linear-map vector commitments and their practical applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 189–219. Springer, 2022. 1.2, 4

[50] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001. 9.3

[51] Ran Canetti, Alley Stoughton, and Mayank Varia. Easyuc: Using easycrypt to mechanize proofs of universally composable security. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 167–16716. IEEE, 2019. 9.3

[52] Megan Chen, Alessandro Chiesa, Tom Gur, Jack O'Connor, and Nicholas Spooner. Proof-carrying data from arithmetized random oracles. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23–27, 2023, Proceedings, Part II*, pages 379–404. Springer, 2023. 9.1.3

[53] Alessandro Chiesa, Michael A Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. *arXiv preprint arXiv:1704.02086*, 2017. 1.1.4, 7.3, 7.2

[54] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable srs. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 738–768. Springer, 2020. 1.1.5, 1.1.6, 1.2, 1.3, 1.3.1

[55] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 769–793. Springer, 2020. 1.1.6, 1.5, **??**, 5.1.5

[56] Arka Rai Choudhuri, Abhihsek Jain, and Zhengzhong Jin. Snargs for p from lwe. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 68–79. IEEE, 2022. 9.1.3

[57] Arka Rai Choudhuri, Sanjam Garg, Aarushi Goel, Sruthi Sekar, and Rohit Sinha. Sublonk: Sublinear prover plonk. *Cryptology ePrint Archive*, 2023. 9.1.2

[58] Heewon Chung, Kyoohyung Han, Chanyang Ju, Myungsun Kim, and Jae Hong Seo. Bulletproofs+: Shorter proofs for a privacy-enhanced distributed ledger. *IEEE Access*, 10:42067–42082, 2022. 1.4

[59] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112, 2012. 1.1.5, 3.5, 6.1.2

[60] Ronald Cramer and Ivan Damgård. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In *Advances in Cryptology—CRYPTO'98: 18th Annual International Cryptology Conference Santa Barbara, California, USA August 23–27, 1998 Proceedings 18*, pages 424–441. Springer, 1998. 1.1.4, 7.3

[61] George Danezis, Cedric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: Building Zerocoin from a succinct pairing-based proof system. In *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*, 2013. 1.1

[62] Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. Weak fiat-shamir attacks on modern proof systems. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 199–216. IEEE, 2023. 8.3.1

[63] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby x. 509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 235–254. IEEE, 2016. 1.1, 1.1.6, 1.2

[64] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1057–1068. IEEE, 2022. 9.1.3

[65] Liam Eagen and Ariel Gabizon. Protogalaxy: Efficient protostar-style folding of multiple instances. Cryptology ePrint Archive, Paper 2023/1106, 2023. 1.1.6, 9.1.1

[66] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 416–426, 1990. 1.1.1

[67] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986. 1.3, 7, 7.1, 7.1, 8.3.1, 8.3.1

[68] Uriel Fiege, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 210–217, 1987. 1.1.1

[69] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Annual International Cryptology Conference*, pages 152–168. Springer, 2005. 1.1.4, 1.2, 7.2, 7.2.1, 7.2.2

[70] Tim Freeman and Frank Pfenning. Refinement types for ML. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, pages 268–277, 1991. 1, 9.2.2

[71] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II 38*, pages 33–62. Springer, 2018. 1.1.3, 1.3

[72] Ariel Gabizon and Zachary J. Williamson. Plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Paper 2020/315, 2020. 9.1.2

[73] 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. 1.1.5, 1.1.6, 1.4, 4.3, 9.1.2, 9.1.2

[74] Joshua Gancher, Kristina Sojakova, Xiong Fan, Elaine Shi, and Greg Morrisett. A core calculus for equational proofs of cryptographic protocols. *Proceedings of the ACM on Programming Languages*, 7(POPL):866–892, 2023. 9.3

[75] Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable SNARKs. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23–27, 2023, Proceedings, Part II*, pages 315–346. Springer, 2023. 7.2.1

[76] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. In *NDSS*, 2014. 1.1

[77] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*, pages 465–482. Springer, 2010. 1.1.5

[78] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pages 626–645. Springer, 2013. 1.1.5, 1.3.1, 1.5.2, 5.1.3, 7.3, 7.3.1

[79] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 99–108, 2011. 1.1.3, 1.3.1

[80] Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo – a Turing-complete STARK-friendly CPU architecture. Cryptology ePrint Archive, 2021. 5.2

[81] Shafi Goldwasser and Yael Tauman Kalai. On the (in) security of the Fiat-Shamir paradigm. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 102–113. IEEE, 2003. 1.1.3

[82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*, pages 173–201. 2019. 9.3

[83] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1), 1989. 1.1.1, 1.2, 7.3, 7.3.2, 9.3

[84] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015. 1.1.5, 3.5

[85] Lorenzo Grassi, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX*, 2020. 5.1.5

[86] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016. 1.1.5, 1.5, **??**, 5.1.5

[87] Mathias Hall-Andersen and Jesper Buus Nielsen. On Valiant's conjecture: impossibility of incrementally verifiable computation from random oracles. In *Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23–27, 2023, Proceedings, Part II*, pages 438–469. Springer, 2023. 5.1.1, 9.1.3

[88] Arend Heyting. Die formalen regeln der intuitionistischen logik. *Sitzungsbericht PreuBische Akademie der Wissenschaften Berlin, physikalisch-mathematische Klasse II*, pages 42–56, 1930. 0

[89] William A Howard et al. The formulae-as-types notion of construction. *To HB Curry: essays on combinatory logic, lambda calculus and formalism*, 44:479–490, 1980. 0

[90] Marvin Jones. Zero-knowledge reductions and confidential arithmetic. 2023. 7.3.2

[91] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. Delegation with updatable unambiguous proofs and PPAD-hardness. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III*, pages 652–673. Springer, 2020. 9.1.3

[92] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010. 4, 7.2.1

[93] Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. MinRoot: candidate sequential function for Ethereum VDF. Cryptology ePrint Archive, Paper 2022/1626, 2022. 4.3

[94] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732, 1992. 1.1.3

[95] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016. 1.2

[96] Abhiram Kothapalli and Bryan Parno. Algebraic reductions of knowledge. In *Annual International Cryptology Conference*, pages 669–701. Springer, 2023. 2, 3, 4

[97] Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. *Cryptology ePrint Archive*, 2022. 1.1.6, 4

[98] Abhiram Kothapalli and Srinath Setty. Cyclefold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. *Cryptology ePrint Archive*, 2023. 9.1

[99] Abhiram Kothapalli and Srinath Setty. Hypernova: Recursive arguments for customizable constraint systems. *Cryptology ePrint Archive*, 2023. 1.1.6, 1.5.2, 4, 4, 5, 5.1.3, 5.2.2, 9.1.2

[100] Abhiram Kothapalli, Elisaweta Masserova, and Bryan Parno. Poppins: A direct construction for asymptotically optimal zkSNARKs. *Cryptology ePrint Archive*, 2020. 1.2, 1.3.1, 1.4, 3.13, 6, 7.3.2

[101] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*, pages 359–388. Springer, 2022. 1.1.6, 1.3.1, 1.3.3, 1.5, 1.5.2, 2.2, 2.3, 4, 4, 5, 5.1.5, 5.1.5, 6, 7.3, 9.1.1, 9.1.2, 9.1.3

[102] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II*, pages 1–34. Springer, 2021. 1.1.5, 1.3.1, 1.4, 4, 6.1.3, 2

[103] Jonathan Lee, Srinath Setty, Justin Thaler, and Riad Wahby. Linear-time zero-knowledge SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/030, 2021. 6.1.3

[104] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. *J. Cryptology*, 16(3), 2003. 1.1.1

[105] Andreas Lochbihler, S Reza Sefidgar, David Basin, and Ueli Maurer. Formalizing constructive cryptography using crypthol. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 152–15214. IEEE, 2019. 9.3

[106] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM (JACM)*, 39(4):859–868, 1992. 1.1.2, 1.4, 3.2, 3.2.1, 3.7, 3.10, 4, 4.3, 4.3, 4.10, 6, 9.1.1

[107] Lurk. https://github.com/lurk-lang. 5.2

[108] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2111–2128, 2019. 1.1.5, 1.1.6

[109] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In *Studies in Logic and the Foundations of Mathematics*, volume 80, pages 73–118. Elsevier, 1975. 9.2.2

[110] Per Martin-Lof. *Intuitionistic type theory*, volume 6. Bibliopolis Naples, 1984. 0

[111] Ueli Maurer. Abstract models of computation in cryptography. In *Cryptography and Coding: 10th IMA International Conference, Cirencester, UK, December 19-21, 2005. Proceedings 10*, pages 1–12. Springer, 2005. 1.1.3

[112] Ueli Maurer. Constructive cryptography–a new paradigm for security definitions and proofs. In *Joint Workshop on Theory of Security and Applications*, pages 33–56. Springer, 2011. 9.3

[113] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987. 1.1.3

[114] Nicolas Mohnblatt. Sangria: a folding scheme for PLONK, 2023. 1.2, 4, 4.3, 9.1.2, 9.2

[115] Wilson Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. Mangrove: A scalable framework for folding-based SNARKs. *Cryptology ePrint Archive*, 2024. 4, 9.2

[116] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *Theory of Cryptography Conference*, pages 222–242. Springer, 2013. 1.1.4

[117] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016. 1.1.4, 1.1.5, 1.5

[118] Rafael Pass. On deniability in the common reference string and random oracle model. In *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*, pages 316–337. Springer, 2003. 1.1.1

[119] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, 1991. 1.4.1, 5.1.5, 7.3.1

[120] Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I*, pages 774–804. Springer, 2021. 1.1.5, 1.2, 1.3.1

[121] Carla Ràfols and Alexandros Zacharakis. Folding schemes with selective verification. Cryptology ePrint Archive, Paper 2022/1576, 2022. 1.2, 1.3.1, 1.3.2, 4, 9.2

[122] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014. 1.1, 1.1.5, 1.1.6, 1.2, 1.5

[123] Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980. 3.2.2, 4.6

[124] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020. 1.1.5, 1.1.6, 1.2, 1.3.1, 1.4, 1.5, 3.5, 4.3, 5.1.5, **??**, 6.1, 6.1.2, 6.1.3, 6.1.3, 3, 7.2, 7.3.2

[125] Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020. 1.5

[126] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with lasso. *Cryptology ePrint Archive*, 2023. 9.1.2

[127] Srinath Setty, Justin Thaler, and Riad Wahby. Customizable constraint systems for succinct arguments. *Cryptology ePrint Archive*, 2023. 4.3, 4.16, 4.3, 9.1.2

[128] Adi Shamir. IP = PSPACE. *Journal of the ACM (JACM)*, 39(4):869–877, 1992. 1.1.2

[129] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16*, pages 256–266. Springer, 1997. 1.1.3

[130] Lev Soukhanov. Reverie: an end-to-end accumulation scheme from cyclefold. *Cryptology ePrint Archive*, 2023. 4

[131] Martin Tompa and Heather Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 472–482. IEEE, 1987. 1.1.1

[132] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath Setty. Transparency dictionaries with succinct proofs of correct operation. In *Network and Distributed System Security (NDSS) 2022*, April 2022. 1.1.6, 6, 6.2

[133] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*, pages 1–18. Springer, 2008. 1.1.6, 1.2, 1.3.1, 1.5, 2.2, 5.1, 5.1.1, 5.2, 9.1

[134] Victor Vu, Srinath Setty, Andrew J Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 223–237. IEEE, 2013. 9.1.2

[135] Riad S Wahby, Ye Ji, Andrew J Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2071–2086, 2017. 1.1.5, 3.5

[136] 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 (SP)*, pages 926–943. IEEE, 2018. 1.1.5, 1.4, 3.5, 6.1.3, 1, 4, 7.3, 7.3.2

[137] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 733–764, Cham, 2019. Springer International Publishing. 1.1.5, 3.5, 6.1.2

[138] Nobuo Yoneda. On ext and exact sequences. *J. Fac. Sci. Univ. Tokyo Sect. I*, 8 (507-576):1960, 1960. 8.4, 8.3, 9.2

[139] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 859–876. IEEE, 2020. 1.1.5, 3.5

[140] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. Integridb: Verifiable sql for outsourced databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1480–1491, 2015. 1.1, 1.2

[141] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vsql: Verifying arbitrary sql queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 863–880. IEEE, 2017. 1.1, 6.1.3

[142] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. A zero-knowledge version of vSQL. Cryptology ePrint Archive, Report 2017/1146, 2017. 1.1, 1.1.6

[143] Zhenfei Zhang. Origami: Fold a plonk for ethereum's vdf. *IACR Cryptol. ePrint Arch.*, 2023:384, 2023. 4.3

[144] Zhichao Zhao and T-H Hubert Chan. How to vote privately using Bitcoin. In *ICICS*, 2015. 1.1

[145] Tianyu Zheng, Shang Gao, Yu Guo, and Bin Xiao. Kilonova: Non-uniform PCD with zero-knowledge property from generic folding schemes. *Cryptology ePrint Archive*, 2023. 4, 9.1.1, 9.2

[146] Zibo Zhou, Zongyang Zhang, and Jin Dong. Proof-carrying data from multi-folding schemes. Cryptology ePrint Archive, Paper 2023/1282, 2023. 1.1.6, 4, 9.2