Amortizing Proofs for Optimal Space Complexity

Joseph Johnston

November 25, 2020

Abstract

Probabilistic proof systems have blossomed into an exciting field, with imminent applications such as a new means for securing blockchains. The foundational concepts of probabilistic proofs used in the early works, including interaction, public vs private randomness, zero-knowledge, substituting multiple provers for hardness assumptions, and algebraic techniques, remain fundamental to the sophisticated proof systems of today. Upon exploring these foundational concepts we explore the particular scenario in which many proofs are to be proven in a batch with minimal space complexity for both the prover and verifier. Our motivation for this scenario is to enable proofs in environments with small space capacity, such as mobile devices and smart contracts on blockchains. We build two constructions of proof systems that allow proving an arbitrary number of proofs with small space complexity for both the prover and verifier. The first utilizes discrete logs, and though not post-quantum secure, is practical with small parameters. The second utilizes lattices, and though less practical with large parameters, is post-quantum secure. Both proof systems are public coin, and the discrete log based proof system is zero-knowledge. Techniques used in our constructions are novel and may be of independent interest. For the discrete log setting, we amortize the evaluation of program-induced polynomials across proofs, only requiring the verifier to perform one final evaluation at the end of the batch. For the lattice setting, we develop a polynomial evaluation technique that recursively randomizes lattice commitments, renormalizes them, checks for consistency between the degenerate and renormalized commitments, and then repeats. Such a recursive technique for lattices provides a promising solution for practical, post-quantum secure proof systems.

Keywords: probabilistic proof systems, zero-knowledge, discrete logs, short integer solution

Contents

Chapter 1

Introduction

1.1 The need for probabilistic proof systems

Proof systems are systems facilitating how a party acting as a prover may prove the truth of a statement to a party acting as a verifier. In our context we focus on proving statements of membership in languages in the NP complexity class, which is defined as the set of languaged in which such membership may be efficiently verified. The prover wishes to prove, and the verifier wishes to verify, that the prover has an NP-witness for an agreed upon NP statement. Sometimes the prover is only tasked with proving that such a witness exists, without necessarily possessing it. When the prover is expected to possess a witness, the proof is called a *proof of knowledge*, and these are the types of proofs we construct in this paper.

To carry out a proof, the prover and verifier interact with each other, sending each other messages while each is unable to see the state of the other. A natural setting is where the prover and verifier operate on electronic devices and exchange messages over the internet. It is fair to say that in every case the prover sends the first message, and the verifier sends the last message. This final message sent by the verifier will be a truth value as to whether the verifier has decided to accept the proof or reject it. The number of *rounds* in the protocol, denoted r for short, is defined as the number of times the prover sends a message and the verifier replies with one. A diagram outlining the generic format of a proof protocol is shown in Figure-1.1.

To make proof systems most efficient, we allow them to be probabilistically correct. Such proof systems are called probabilistic proof systems. Two properties must be satisfied by a probabilistic proof system.

- Completeness: If the prover has a valid witness and complies with the protocol, the verifier will accept.
- Soundness: If the prover does not have a valid witness or does not comply with the protocol, the verifier rejects with high probability.

Note that soundess yet not completeness is probabilistic. This can be adjusted, but most natural protocol constructions and those of interest in this paper agree with this difference.

The simplest example of a probabilistic proof protocol is the following:

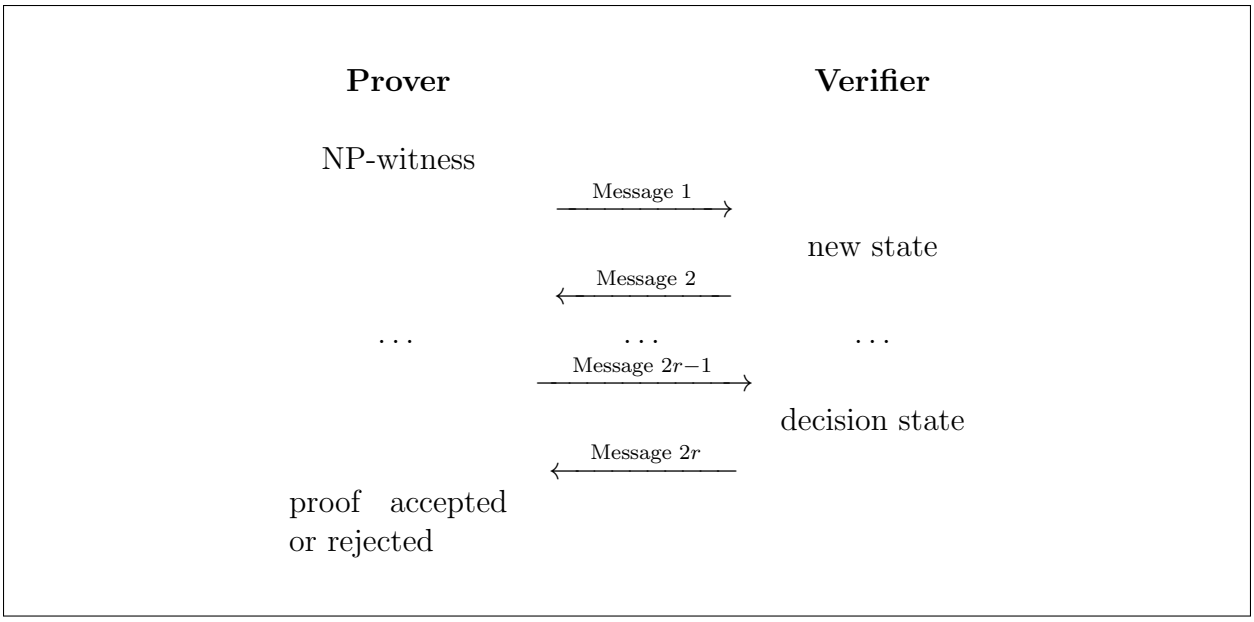

Figure 1.1: Prover and verifier interactive protocol

1. The prover sends the NP-witness to the verifier.
2. The verifier checks whether the witness is valid in polynomial time with respect to the statement size, and sends back 'accept' or 'reject' accordingly.

This protocol has only one round, and satisfies both completeness and soundeness. But this example is supremely uninteresting, not least because the soundness is not probabilistic. What makes probabilistic proof systems of any interest is that we avoid sending the entire witness to the verifier. It is instead desirable the prover only send small snippets of the witness or metadata to significantly reduce the space complexity burden on the verifier. With less data to analyze by the verifier, the time complexity burden on the verifier will also improve.

There is another feature of probabilistic proof systems that we find of interest. This feature, known as *zero-knowledge*, is when the verifier learns nothing about the prover's witness, yet is still able to accept or reject with appropriate completeness and soundeness respectively. The above protocol example is clearly not zero-knowledge as the verifier sees the prover's witness in its entirety.

1.2 Notions of proofs and knowledge

The word 'proof' is usually taken to be a noun. In Merriam-Webster Dictionary (`merriam-webster.com/dictionary/proof`), the first meaning of 'proof' listed (and conveying the earliest historical meaning) reads as follows.

> the cogency of evidence that compels acceptance by the mind of a truth or a fact

A proof is evidence that compels one to accept whatever is to be proven. The definition does not restrict the possible forms of the evidence. Evidence can take either a dynamic or a static form. In dynamic form, a conversation in which one person convinces another could serve as evidence. In static form, a certificate from a trusted party could serve as evidence. In the case of dynamic evidence, the proof is the dynamic process conveying the evidence. In the case of static evidence, the proof is the static object containing the evidence. When the evidence is communicated dynamically, we say the proof is *interactive*. When the evidence is communicated statically, we say the proof is *non-interactive*. A proof system is interactive or non-interactive depending on whether it features interactive or non-interactive proofs.

The standard practice of implementation within the crypto community treats proofs as static blobs of data that computers read and write. A proving computer writes a non-interactive proof and sends it to a verifying computer that reads it. In theory, however, proofs are more often designed and analyzed as dynamic processes of communication between computers. A proving computer and verifying computer interact by sequentially sending messages between each other. Regardless whether a proof system is interactive or non-interactive, there always exists a proving party, referred to as the *prover*, and a verifying party, referred to as the *verifier*. The purpose of any proof system is for the prover to convince the verifier that a given statement is true. The prover claims the statement is true, but the verifier may be doubtful, so the prover must prove the statement.

There exists a plethora of features a proving system can satisfy, such as all kinds of practicality measures which are of tremendous importance for our purposes. Another umbrella of features exist in regard to privacy, in particular regarding what exactly the prover must share with the verifier in order to prove the statement. By proving the statement, the prover inevitably reveals to the verifier that the statement is true. If the verifier trusts the prover, all that the prover needs to share is this single bit of information that the statement is true. In the default case that the verifier does not trust the prover, however, more information must be shared in order for the proof to be convincing. Privacy for the prover may be preserved if the information shared reveals nothing the prover is afraid to share. To be more strict, one may also require that the information shared be useless to the verifier for anything beyond verifying the proof. In terms of computation, information is useful when it can be used to compute something useful. Thus in our setting where proofs are exchanged between computers, we may require that the verifier cannot use the information to compute anything useful beyond verification. The latter requirement is informally what stands behind the formal definition of 'zero-knowledge.'

1.3 Zero-knowledge through the simulation paradigm

Knowledge is information that is useful for computation. For example, a bit-string representing a satisfying assignment to a certain boolean circuit may be useful. On the other hand, a bit-string representing the outcomes of coin tosses may not be useful. In the context of proof systems we are interested in the verifier's knowledge and how it changes before and after the proof. Zero-knowledge means the verifier's knowledge remains the same before and after the proof, that is the verifier gains zero knowledge through the proof regardless how the proof is communicated, whether interactively or non-interactively. For example, suppose the

prover reveals a bit-string to the verifier. If the bit-string represents a satisfying assignment to a certain boolean circuit, the verifier has gained knowledge if that satisfying assignment was not previously known to the verifier. On the other hand, if the bit-string represents coin tosses, the verifier has not gained knowledge because the verifier could have tossed the coins itself without any help from the prover.

The notion of knowledge as useful information is difficult to define precisely. With no precise definition of knowledge, the quantification of knowledge (known as *knowledge complexity*) is also difficult to define. Due to lack of any natural and convenient way to define and quantify knowledge, zero-knowledge has always been defined in a conservative manner that avoids the need for any precise notions regarding knowledge. It may seem ironic that we can define a quantity of knowledge, namely zero knowledge, without any quantification system for knowledge. But the quantity zero is special and different from any other quantity. The nature of any non-zero quantity varies with the quantification system, but the nature of zero (in essence, 'nothing') remains invariant regardless of the quantification system. Thus we may define the quantity zero-knowledge without having in mind any quantification system for knowledge.

To guarantee that the prover shares zero knowledge with the verifier, we require that whatever the prover shares with the verifier could have been generated by the verifier itself with no access to the prover. How do we define 'whatever' the prover shares with the verifier, and once defined what does it mean for the verifier to 'generate' it? Roughly, every prover gives rise to a family of random variables describing the proofs or some function of the proofs, such as a function the verifier may wish to compute when given a proof. We require that the verifier *simulate* the family of random variables to the extent that sampling from the verifier's simulated random variables is indistinguishable from sampling from the prover's real random variables. In other words, every prover gives rise to a distribution ensemble, and we require the verifier to simulate a fake distribution ensemble indistinguishable from the real one. There are varying degrees of indistinguishability giving rise to different degrees of zero-knowledge security. The two distribution ensembles may be identical, statistically close, or computationally close, resulting is perfect zero-knowledge, statistical (or almost perfect) zero-knowledge, or computational zero-knowledge respectively. By convention, the unqualified definition of zero-knowledge is taken to be computational zero-knowledge.

1.4 On the need for mathematics

Mathematics underpins proof systems such as the two in this porject because it offers a well established framework for rigorous analysis necessary to justify security. Here we are referring to the use of mathematics in the *construction* of proof systems, but other types of mathematics are used in the *analysis* of proof systems. Here we discuss the motivations behind the use of finite algebraic structures (in particular rings and fields) throughout this project and other proof systems.

1.4.1 Restricting to finite structures

While mathematical structures are many and varied, only those suited for constructing proof systems are of interest for our purposes. What exactly constitutes suitability for proof systems is not yet fully qualified, but a minimal requirement is efficient representation and manipulation on a computer. Infinite structures can be challenging to handle in this sense, especially when enforcing consistency across computers, which is crucial in the case of proof systems involving many parties. Therefore, narrowing focus to finite structures seems justifiable.

By many metrics, the majority of mathematical research targets infinite structures, so narrowing focus to finite structures excludes the majority of possibilities. This may be seen as both a boon and a bane in that while limiting opportunity it also focuses attention. The rich class of structures just beyond finite structures is finitely *generated* structures, which cannot be entirely ruled out, and as such the choice of finite structures is not fully justified. Nevertheless, finite structures are the structures of choice for now.

1.4.2 Choice of finite structures

Restricting to finite structures still leaves an infinite number of possibilities. We must further restrict attention to those structures in which we see opportunity for proof systems. In fact, we must restrict even further to those that seem appropriate for *practical* proof systems.

Diverging for a moment, the term 'practical' for qualifying proof systems, or more generally any solution to a technical problem, can be ambiguous. Some solutions are capable of implementation in the near future, say the next decade. These solutions are largely dependent on current technologies and their near-term trajectories. Other solutions are only capable of implementation some time in the distant future. These solutions are largely insensitive to the current state of technology. Throughout this project we refer to the former types of solutions as 'practical' and the latter types as 'impractical'. In both cases, academic literature may refer to a solution as 'efficient' because it requires bounded resources as a function of the problem size. Specifically, bounds are taken to be polynomial functions in order that efficient solutions may be combined in various ways while remaining efficient. Do not confuse 'efficient' with 'practical' as many solutions are efficient but impractical. A solution may be efficient since it meets certain resource bounds, and yet impractical since the resources demanded are more than current technology can provide.

In our context the solutions we seek are proof systems, and the technologies of concern are computing and networking technologies. In particular, we are concerned with the affordable speed and volume of computer calculations and network signals. These metrics are what separate the practical from the impractical.

Academic literature on proof systems can be roughly split into the 'theoretical' type and the 'applied' type. Theoretical research is motivated by efficient solutions, which may be impractical. Applied research is motivated by practical solutions, which are always efficient. Since theoretical research pursues solutions independent of current technology, any mathematical structure amenable for proof systems suffices. Since applied research pursues solutions highly dependent on current technology, choices of mathematical structures are more severely limited.

A set of finite structures of popular use in theoretical research but of rare use in applied research is graphs, that is the set of structures each consisting of a finite number of objects and the connections between them. For example, expander graphs played a central role in [Din06] to prove the PCP Theorem, likely the most important result in theoretical proof systems. Graphs are extensively studied, including the expander graphs that may be of use for our purposes. The use of graphs in proof systems, however, seems to require resources beyond the capacity of modern technology.

On the other hand, a set of finite structures of popular use in both theoretical and applied research is finite fields, that is a certain set of highly symmetric algebraic structures. Finite fields are extensively studied and possess many homomorphic properties desirable for our purposes. Furthermore, many uses of finite fields in proof systems require resources within the capacity of modern computing and networking technologies. Finite fields are the finite objects of study in 'field theory' which itself is embedded in 'ring theory.' Finite fields are the basis for nearly all proof systems used in both theory and practice today, and they indeed underpin both proof system constructions we provide in this project.

Chapter 2

Literature Review

2.1 The founding papers of interactive proof systems

In this project we are interested only in proof systems of probabilistic correctness because guaranteed correctness leaves no opportunity for the features we seek. Thus by 'proof systems' we mean 'probabilistic proof systems.'

2.1.1 Introduction

Two works are usually regarded as the primary founding papers of (probabilistic) proof systems. The first paper is by Goldwasser, Micali, and Rackoff [GMR85] and doesn't have a precise publishing date because early versions of the paper circulated as early as 1982, but it was rejected from major journals three times before being accepted into *STOC'85* more than a decade later [Gol02]. The second paper is by Babai [Bab85] and appeared in 1985 and was in fact presented at the same conference (*STOC'85*) as [GMR85]. The two works were independently created, but given their simultaneous publishing and related topics each work cited the other.

We note that [GMR85] was also the founding paper of the concept of zero-knowledge, more generally knowledge complexity. After introducing interacting proofs, [GMR85] took a natural next step and considered how much knowledge a proving process reveals to the verifier beyond the validity of the statement. The concept of knowledge complexity, however, can be examined separately and we do not examine it here.

The two works independently define their own basic tools, proof systems employing these tools, and complexity classes describing languages in terms of these proof systems. The resulting proof systems and complexity classes carry both similarities and differences. A subsequent paper [GS86] proved the complexity classes are in fact equivalent.

2.1.2 The framework of GMR85

Basic Tools

The tools defined in [GMR85] are pairs of interactive Turing machines. Both machines read the same input tape. Each machine has two private tapes, a random tape for reading, and

a work tape for reading and writing. Additionally there are two tapes for communication between the machines. Each communication tape facilitates communication in one direction by one machine writing to it and the other reading from it. They contrast this model of communicating Turing machines with the model for the traditional NP proving process in which the random tapes are nonexistent and there is only one communication tape from the prover to the verifier.

Proof systems

To convert a pair of interactive Turing machines into a proof system for a language L, [GMR85] treats one machine as the prover and the other as the verifier and enforces the following conditions.

- Unbounded prover The prover has unbounded computational resources.
- Bounded verifier The verifier is bounded by polynomial time.
- Completeness For every $x \in L$, the verifier halts and accepts with probability at least $1 - 1/p(|x|)$ for all polynomials p.
- Soundness For every $x \notin L$, the verifier halts and accepts with probability at most $1/p(|x|)$ for all polynomials p.

In this case we say L *has* a proof system.

Complexity classes

With a definition of proof systems in hand, [GMR85] defines induced complexity classes. They define the complexity class *Interactive Polynomial-time*, denoted IP, as containing all languages that have a proof system as previously defined. Any such proof system is referred to as an interactive proof system. The authors believe the number of messages exchanged between the two machines, capturing the amount of interaction required, is the primary metric for efficiency. In light of this they further partition IP into subclasses according to how the amount of interaction grows with the input size. The class $\mathcal{IP}[t(\cdot)]$ for $t\colon \mathbb{N} \to \mathbb{N}$ comprises languages having a proof system with the exchange of $t(|x|)$ messages on instance x. To make this precise, [GMR85] assumes the verifier sends the first message.

2.1.3 The framework of Bab85

Basic Tools

The tools defined in [Bab85] are *Arthur vs. Merlin games* or Arthur-Merlin games. Arthur-Merlin games are a special case of *Games against Nature* from [Pap84]. A Game against Nature, in turn, is a special case of a more general game involving two players on a common input that sequentially exchange an prespecified number of messages, and at the end a deterministic polynomial time Turing machine views all moves taken and declares a winner. A Game against Nature is when one of the players, called Nature, is indifferent to winning and on each move simply sends a random message. An Arthur-Merlin game is a Game

against Nature in which Arthur plays the role of Nature and for any input the probability of Merlin winning is bounded away from one-half in one direction or the other (e.g. probability less than 1/3 or greater than 2/3). The purpose of the latter condition will become apparent when using Arthur-Merlin games as proof systems.

Proof systems

While Arthur-Merlin games may seem irrelevant to proof systems they may in fact serve as proof systems. The prover can be identified with the powerful wizard Merlin, appropriately so in that Arthur-Merlin games impose no computational restrictions on Merlin. The verifier can be identified as the machine that makes the random moves of Arthur and then invokes the deciding Turing machine at the end. Thus the verifier is restricted to only asking random questions and once all answers are received to making a decision in deterministic polynomial time. If we invoke this proof system on a language containing exactly those strings on which Merlin wins with probability more than half, then the final condition of Arthur-Merlin games promises notions of completeness and soundness analogous to those of [GMR85]. In order to guarantee completeness and soundness, Arthur-Merlin proof systems are appropriate for exactly such languages. It may seem backwards that we first define Arthur-Merlin proof systems and then seek appropriate languages, but this is an artifact of how [Bab85] organizes definitions. In practice we first define languages and then seek appropriate Arthur-Merlin proof systems. Either way, the same pairs of languages and proof systems exist.

Complexity classes

With Arthur-Merlin games in hand, [Bab85] defines induced complexity classes. Every Arthur-Merlin game induces a language consisting of all inputs for which Merlin succeeds with probability above one half. Thus we may define sets of languages, that is complexity classes, by defining sets of Arthur-Merlin games. Suppose we partition Arthur-Merlin games by which player moves first and also by how many moves take place. Let $t\colon \mathbb{N} \to \mathbb{N}$ be a function of game input lengths. For games where Arthur moves first and there are $t(\cdot)$ moves, denote the corresponding complexity class by $\mathcal{AM}[t(\cdot)]$. For games where Merlin moves first and there are $t(\cdot)$ moves, denote the corresponding complexity class by $\mathcal{MA}[t(\cdot)]$. In the case that there are a constant number of moves, we may omit the brackets and instead expand the capital string to indicate the sequence of moves. For example, the complexity classes induced by single-move games may be denoted A and M, two-move games AM and MA, three-move games AMA and MAM, etc.

2.1.4 Similarities

Both forms of (probabilistic) proof systems rely crucially on *interaction*, in particular non-trivial interaction beyond the prover simply sending the verifier a proof. The interaction of [GMR85] takes place via interacting Turing machines, while the interaction of [Bab85] takes place via a game and a sequence of moves between two parties. All proof systems that have followed these works (including MIPs, PCPs, NIZKs) still employ some form of non-trivial interaction, even if the verifier only interacts once with the prover.

Both forms of proof systems employ a randomized verifier. In contrast, verifiers for the NP proof system are deterministic. The need for verifier randomization is equally important to the need for verifier interaction. In fact, in most proof systems including all those explored in this project, the randomization and interaction of the verifier can be thought of as the same. They are the same in that all randomness is only used in interaction, and all interaction only uses randomness. This is how verifiers function in the proof systems of [Bab85], whereas verifiers function more freely in the proof systems of [GMR85].

In both forms of proof systems the prover has unbounded computational resources and the verifier is restricted to probabilistic polynomial time (and [Bab85] further restricts exactly how the randomness is used). It seems both papers arrived at this asymmetric formulation for the same reasons. If the prover were restricted, the prover could not prove many statements otherwise provable. If the verifier were unrestricted, the verifier could prove many statements to itself with no need for a prover. Both works intend to capture the theoretical power of interactive proof systems with no intention for practicality. Works that follow, such as this project, narrow focus to proof systems with probabilistic polynomial time provers.

2.1.5 Differences

The primary difference between the proof systems of [Bab85] and [GMR85] arises due to the already mentioned difference of how [Bab85] restricts the verifier's use of randomness. The verifier of [GMR85] has a private random tape and may thus hide randomness from the prover. The verifier of [Bab85], on the other hand, only may use the randomness of Arthur all of which is sent to the prover. A useful analogy is drawn by [Bab85] wherein interactive proofs from [GMR85] are card games (in which cards may be private) and interactive proofs from [Bab85] are chess games (in which all moves are public). Interactive proofs with private randomness are qualified as 'private coin' whereas those with public randomness are qualified as 'public coin'.

The notation of complexity classes also differs between the two works. The two notations $\mathcal{IP}[t(\cdot)]$ and $\mathcal{AM}[t(\cdot)]$ (and $\mathcal{MA}[t(\cdot)]$) are similar in that they both involve brackets indicating the number of messages exchanged. But when we omit the brackets they suddenly carry contrasting meanings. IP captures languages with interactive proofs of any polynomial number of messages, while AM (and MA) captures languages with interactive proofs of only two messages. Thus with brackets omitted IP allows a maximum number of bidirectional messages, while AM (and MA) allows a minimum number of bidirectional messages.

We also mention that the completeness and soundness probabilities are presented differently but they are effectively equivalent. The error probabilities in [GMR85] are required to be negligible, whereas the error probabilities in [GMR85] may not be negligible but they are formulated such that they will become negligible upon repeating the protocol any non-constant polynomial number of times.

2.1.6 Analysis and equivalence

The complexity classes formulated by [Bab85] are more amenable to analysis than those of [GMR85] due to the public coin restriction [Bab85] imposes on the verifier. While [GMR85]

provides no analysis of the complexity classes $\mathcal{IP}[\cdot]$, [Bab85] makes several trivial and nontrivial observations about $\mathcal{AM}[\cdot]$ and $\mathcal{MA}[\cdot]$. Many papers followed making further observations about these classes and their relationships to each other and to other classes.

Trivially, [Bab85] notes A = BPP and M = NP. To see why A = BPP, see that Arthur tosses coins and Merlin sends nothing, so a deterministic polynomial time machine is left to decide membership only with the help of Arthur's coins. To see why M = NP, see that Merlin sends the equivalent of an NP witness and Arthur sends nothing, so a deterministic polynomial time machine is left to decide membership only with the help of Merlin's witness. It is also trivial to note the inclusion relation relative to the number of messages exchanged which holds for the IP classes as well. In particular, if a language has a proof system with k messages exchanged then it also has a proof system with $k' > k$messages exchanged where the additional messages are empty. Extending the inclusion relation by concatenating moves also at the beginning of the game we have

$$\mathcal{AM}[k] \cup \mathcal{MA}[k] \subseteq \mathcal{AM}[k+1] \cap \mathcal{MA}[k+1] \tag{2.1}$$

Nontrivially, [Bab85] proves that an MA game can be simulated by an AM game with only a polynomial increase in the total size of messages sent, and thus any constant length game can be simulated by a single AM game. Reversing the moves of an MA game to obtain an AM game lets Merlin adaptively choose his message after learning the randomness of Arthur which is problematic for soundness. To overcome this bias in Merlin's message, [Bab85] plays the AM game many times in parallel requiring Merlin to reply the same to each. The result is a single AM game with polynomially larger messages. One may then take any Arthur-Merlin game and make a constant number of MA to AM swaps and then merge adjacent A's and M's all while maintaining polynomial message sizes. Consequently all finite levels of the hierarchy above AM collapse down to AM, that is for any constant k

$$\mathcal{AM}[k] = \mathcal{AM}[2] \tag{2.2}$$

Unbounded levels of the hierarchy specified by any non-constant polynomial, however, are not known to collapse.

Following [GMR85] and [Bab85], [GS86] proves that private coin interactive proofs can be simulated by public coin interactive proofs with the same number of messages but with one additional message sent from Merlin to Arthur at the beginning. In particular for any polynomial t they prove

$$\mathcal{IP}[t(\cdot)] \subseteq \mathcal{MA}[t(\cdot)+1] \subseteq \mathcal{AM}[t(\cdot)+2] \tag{2.3}$$

Combined with the collapsing theorem of [Bab85] one may collapse the additional two messages of the AM game into the polynomial t. Combining further with the trivial fact that proof systems from [Bab85] are special cases of proof systems from [GMR85] we obtain the equality

$$\mathcal{IP}[t(\cdot)] = \mathcal{AM}[t(\cdot)] \tag{2.4}$$

Thus the two forms of proof systems have equal power for proving statements, and private verifier randomness provides no additional power over public verifier randomness. We warn,

however, that this does not immediately imply equivalence with respect to any feature of proof systems (e.g. zero-knowledge).

The power of interactive proofs was well underestimated in these early days. An early extended abstract of [GMR85] conjectured that AM is a strict subset of $\mathcal{IP}[2]$, and that the IP hierarchy doesn't collapse, that is $\mathcal{IP}[k]$ is a strict subset of $\mathcal{IP}[k+1]$. Given the above equivalence and collapsing theorem, both of these conjectures turned out to be false as $\mathcal{AM} = \mathcal{IP}[2]$ and the finite IP hierarchy collapses. On the other hand, [Bab85] describes the family of languages having Arthur-Merlin proof systems as "just above NP" believing it is not much larger than NP, and such belief was not unique to [Bab85]. The Arthur-Merlin games were inspired by the Games against Nature of [Pap84] with the added constraint that winning and losing probabilities against Nature are bounded away from one-half. Though [Pap84] showed Games against Nature characterize PSPACE, [Bab85] believed Arthur-Merlin games characterize a much smaller complexity class. In particular, [Bab85] believed that even coNP could not exist within $\mathcal{AM}[\text{poly}(\cdot)]$, but later [LFKN92] proved to the contrary that $\rfloor\!\lfloor\mathcal{NP} \subseteq \mathcal{IP} = \mathcal{AM}[\text{poly}(\cdot)]$. Ultimately it was proven by [Sha92] that interactive proof systems characterize PSPACE. In other words, languages with interactive proof systems are equivalent to languages computable in polynomial space, that is

$$\mathcal{PSPACE} = \mathcal{IP} = \mathcal{AM}[\text{poly}(\cdot)] \tag{2.5}$$

This result highlights how interactive proof systems are believed to be much more powerful than NP proof systems.

2.1.7 Motivations and results

The founding works [GMR85] and [Bab85] each arrive at interactive proof systems by their own motivations and each arrive at their own corresponding results. In [GMR85] attention is drawn to interactive proof systems because they seem efficient by intuition and because they offer the ability to prove statements in zero-knowledge in contrast to previous nontrivial proving systems. They construct zero-knowledge interactive proofs for quadratic residuosity and quadratic nonresiduosity. If it were not for these zero-knowledge proofs the paper would not have presented anything about interactive proof systems provably superior to previous proof systems. We also note that the zero-knowledge proofs they present rely on a non-deterministic prover and thus do not suffice for efficient protocols. In [Bab85] attention is drawn to interactive proof systems because they allow proving certain statements from group theory either not known to be provable in the NP proof system or rather complex to prove in the NP proof system. In particular, the problems are deciding membership and non-membership in a group and deciding order and non-order of a group when groups are matrix groups represented by generating sets. A particular version of the membership problem (4 by 4 integral matrices) is even known to be undecidable, and yet [Bab85] shows it may be proven probabilistically via interactive proofs.

Each work also comments philosophically on the significance of interactive proofs. The authors of [GMR85] emphasize the interactivity, contrasting how NP proofs are analogous to those that can be "written down in a book," whereas interactive proofs are analogous to those that can be "explained in class." A proof written down in a book must answer all

possible questions, whereas a proof explained in class must only answer questions as they arise. The author of [Bab85] emphasizes randomness, saying "a random string can sometimes replace the most formidable mathematical hypothesis" and recalling how randomness often radically simplifies solutions, the classic example being primality testing. The philosophy of [Bab85] is contained in its title, "Trading group theory for randomness," referring to a trade off that statements not easily proven via group theory can be easily proven via randomness if one is willing to trade complete soundness for statistical soundness. Lastly, we note that the class AM has evidence for being a natural class. As [Bab85] points out, one may prove AM relates to NP in the same naturally relativized way that BPP relates to P. In particular, for a random oracle O (meaning for almost every oracle O in a measure-theoretic sense)

$$\mathcal{BPP} = \mathcal{P}^O \text{ and } \mathcal{AM} = \mathcal{NP}^O \tag{2.6}$$

Thus AM might be thought of as a probabilistic version of NP in the same way BPP is a probabilistic version of P. At the same time, MA might also be thought of as a probabilistic version of NP in which the NP verifier may use randomness. Perhaps both AM and MA are indeed deserving of the name "just above NP."

2.2 Private-coin vs public-coin proofs

The work of [GS86] proves the two founding forms of proof systems are equivalent in that anything provable in one is provable in the other. The two forms of proof system were introduced independently by [GMR85] and [Bab85] with the primary difference that the verifiers of [GMR85] may hide private randomness from the prover, while the verifiers of [Bab85] must reveal all randomness to the prover. Proof systems allowing private verifier randomness are referred to as private-coin proof systems, and those not allowing it as public-coin proof systems. One may immediately observe that any public-coin proof system is a private-coin proof system in which no randomness in hidden. Conversely, the paper proves that any private-coin proof system can be simulated and thus replaced by a public-coin proof system with roughly the same efficiency.

Here we present how the work of [GS86] showed that a private coin protocol can be simulated by a public coin protocol. For ease of notation we borrow from [Bab85] and refer to the public-coin verifier and prover as A and M for Arthur and Merlin respectively. We warn the reader not to confuse this notation with that for the complexity classes of [Bab85]. The prover M and verifier A of the public-coin protocol simulate the prover P and verifier V of the private-coin protocol. In effect M proves to A that P could have proven the original statement to V.

Suppose we assume P is deterministic by always using the optimal strategy to convince V. Suppose we fix the statement to be proven thus leaving the whole protocol, including all messages sent therein and the final decision, a deterministic function of the private randomness of V. In other words, for every (private) random input string of V a unique protocol ensues. The probability of P convincing V is equal to the fraction of the random strings inducing protocols that result in acceptance. We wish for M to prove to A that the set of random strings resulting in acceptance is adequately large. First we introduce a tree structure to describe this set in terms of smaller sets. Second we describe the core subprotocol

of [GS86] and outline how it is used to recursively prove the size of sets in terms of smaller sets. Last we cover crucial details omitted in the outline.

2.2.1 The acceptance tree

Build a tree structure to represent all possible message paths (sequences of messages) that V may take resulting in acceptance. Each edge can be identified with a message from V and each vertex can be identified with the unique message path leading to it from the root. We also identify each vertex with a unique set of random strings. Specifically, a random string r is in the set identified with a vertex v if and only if the message path of Vpasses through v when V uses r as its random input. The root vertex corresponds to the empty path and thus to the set of all random strings resulting in acceptance. It is this set which M must prove to A has adequate size, and we refer to it as the root set. Each leaf vertex corresponds to a message path resulting in acceptance and thus to the set of all random strings inducing this acceptance path. We refer to sets associated with leaf vertices as leaf sets. For all instances of parent and children vertices we refer to their associated sets as parent and children sets.

One may observe that each parent set is the union of the children sets, and furthermore the children sets are a partition of the parent set. The partition relationship exists between parent and children sets because each random string induces a unique message path, so no random string in a given parent set can be inherited by more than one child set. Due to the partition relationship the size of any parent set can be measured as the sum of the sizes of its children sets.

The statement to be proven in the public-coin protocol is a lower bound on the size of the root set. To measure the root set, we could recursively measure all child sets. Summing the size of all leaf sets, however, seems to require many more messages to be exchanged than in the private-coin protocol. To keep the number of messages in the public-coin protocol roughly the same, the technique of [GS86] proceeds differently by approximating a lower bound on the size of the root set.

2.2.2 Protocol outline

Before outlining the protocol we introduce the core subprotocol of [GS86] in slightly abused form for the sake of simplicity. The subprotocol reduces proving a lower bound on the size of a subset σ (of some superset) to proving membership in σ. The verifier randomly chooses another subset ρ in which the verifier can efficiently check membership. The size of ρ should be proportional to the size of σ (so the prover may need to send auxiliary information to help the verifier select a size for ρ). Upon receiving ρ from the verifier, the prover should be able to find with high probability an element x belonging to both ρ and σ if and only if $|\sigma|$ meets the claimed lower bound. Upon receiving x from the prover, the verifier efficiently checks membership of x in ρ. Thus we have reduce to proving membership of x in σ. The verifier accepts the claimed lower bound on $|\sigma|$ if and only if both memberships hold. In [GS86] the random set is chosen as the preimage of a random target set with respect to a random hash function, such that verifying membership entails efficiently hashing and examining the output with respect to the target set.

Using this subprotocol, [GS86] reduces proving lower bounds on the size of any non-leaf set (such as the root set) to proving lower bounds on the size of a child set. We proceed recursively from the root set to a child set, continuing until reaching a leaf set, at which point we invoke the suprotocol again to prove a lower bound on the leaf set. For simplicity we focus on an arbitrary non-leaf set Σ. To prove a lower bound on $|\Sigma|$ we prove that at least $\texttt{count}_\Sigma$ children sets have size at least $\texttt{size}_\Sigma$. Let σ_Σ be the subset of children sets each of size at least $\texttt{size}_\Sigma$. We wish for M to prove to A that $|\sigma_\Sigma| \geq \texttt{count}_\Sigma$, thus proving the lower bound $|\Sigma| \geq \texttt{count}_\Sigma \times \texttt{size}_\Sigma$. Using the subprotocol, A sends a random challenge to M and M replies with a child set x leaving A to verify $x \in \sigma_\Sigma$, that is $|x| \geq \texttt{size}_\Sigma$. If x is not a leaf set we proceed recursively as above proving the lower bound on $|x|$. If x is a leaf set we invoke the subprotocol again to reduce verifying the lower bound on $|x|$ to verifying membership in x. Since x is a leaf set consisting of arbitrary random strings, any string may be accepted as a member of x.

2.2.3 Omitted details

The protocol outline above omits crucial details and oversimplifies perhaps to the point of confusion. The prover M doesn't send child sets as said, but instead sends pairs of messages. The first is the next message V would have sent, and the second is the next message P would have sent in response. The message from V serves as an identifier for the child set, and the subprotocol actually hashes the message rather than the associated child set. The message from P is not used immediately by A but is saved and accumulated into a transcript with all other messages between P and V. When M finally sends the random string in the leaf set, A saves it as the random input for V. To decide whether to accept or reject the statement, A delegates the decision to V by invoking it on the original statement, the saved random input, and the message transcript.

This public-coin protocol is indeed a simulation of the private-coin protocol because every interaction of A and M simulates an interaction of P and A. Specifically, every time A sends a challenge M replies with a question and answer pair from V and P. Actually M also replies with some auxiliary information concerning how A should select its next challenge (in particular how large the random set ρ should be). The need for M to send auxiliary information for A to select its next challenge explains why the public-coin simulation requires an extra message at the beginning from M. In order to be consistent in having the verifier send the first message, we may say A sends an initial empty message followed by the auxiliary information from M. In this case the public-coin protocol has two more messages than the private-coin protocol.

The work of [GS86] also presents a variant of Turing machines that characterize the same languages as public-coin proof systems. They refer to the machines as probabilistic, nondeterministic, polynomial time Turing machines because they run in polynomial time and may select the next configuration either deterministically or nondeterministically in one of two ways. The first way is choosing the next configuration randomly, analogous to a verifier sending a random message. The second way is choosing the next configuration to maximize the probability of acceptance, analogous to a prover sending an optimal response.

2.3 Multi-prover interative proofs

Multi-prover interactive proofs (MIPs) introduced by [BOGKW88] are a modification of the original form of interactive proofs introduced by [GMR85]. Instead of a single all-powerful prover there are multiple all-powerful provers who may not communicate during the protocol. If a language has a multi-prover interactive proof it is in the complexity class MIP. Regular interactive proofs are a special case of multi-prover interactive proofs, so $\mathcal{IP} \subseteq \mathcal{MIP}$. This containment is believed to be strict because allowing multiple provers seems to significantly increase the power of the proof systems as far as what languages can be proven. Specifically, analogous to how [Sha92] proved that $\mathcal{IP} = \mathcal{PSPACE}$, later [BFL91] proved that $\mathcal{MIP} = \mathcal{NEXP}$, and indeed the containment $\mathcal{PSPACE} \subseteq \mathcal{NEXP}$ is believed to be strict.

2.3.1 Motivation

The motivation of [BOGKW88] for introducing MIPs was to find an efficient proof system for NP with perfect zero-knowledge. Perfect zero-knowledge with unbounded provers leaves no room for error in that the prover uses the optimal strategy and the verifier learns nothing in the information-theoretic sense. These properties were exploited in [For87] to show if a language can be proven with an unbounded verifier in perfect zero-knowledge, then so can its complement language. From this [For87] concludes that NP must have no perfect zero-knowledge proof system, and indeed regular interactive proof systems for NP with perfect zero-knowledge are only known to exist assuming a computationally bounded prover.

Wishing to avoid intractability assumptions regarding the proving party, [BOGKW88] sought alternative assumptions regarding the proving party. The alternative assumption put forth by [BOGKW88] is that multiple independent provers assist the verifier. By independent we mean the provers do not communicate among each other during the proving process, because if they did they would effectively become a single prover. While the work of [BOGKW88] does indeed achieve its goal of presenting an efficient proof system for NP with perfect zero-knowledge, the notion of MIPs in and of itself was a major contribution inspiring much subsequent research.

2.3.2 Additional Results

In addition to introducing MIPs and showing it suffices for perfect zero-knowledge for NP, [BOGKW88] makes two additional important observations. Other works followed proving further observations.

The first observation is that more than two provers are unnecessary.

Proposition 1 (Two provers are sufficient for MIPs [BOGKW88])**.** *As proven in [BOGKW88] any k provers can be replaced by just two provers.*

Proof. Instead of interacting with the k provers the verifier interacts with two provers as follows.

1. The verifier sends all its coins to the first prover and asks for the transcript of what would have occurred if it had interacted with the k provers using these coins. The transcript consists of k sub-transcripts, sub-transcript $i \in [k]$ being the transcript between the prover i and the verifier.

2. The first prover replies with a potentially erroneous transcript. If the transcript is correct then the verifier can make a correct decision based on whether or not it would have accepted had it interacted with the k provers. But the first prover may have sent an erroneous transcript, so the verifier interacts with the second prover to determine if this is so.

3. For each $i \in [k]$ the verifier interacts with the second prover executing the protocol it would have executed with prover i. But instead of executing the subprotocols in some predetermined order the verifier randomizes their order. Upon completing each interaction the verifier compares the transcript of the interaction with the relevant sub-transcript sent from the first prover, expecting them to match. If the verifier ever encounters a mismatch it rejects. Otherwise the verifier accepts.

Completeness evidently holds. Soundness holds because at if the original verifier would have rejected then at least one of the sub-transcript sent by the first verifier is erroneous. The second verifier must match the erroneous sub-transcript by using the same erroneous strategy when interacting with the verifier. But there are many possible erroneous strategies and without seeing the verifier's coins the second prover doesn't know which erroneous strategy the first prover chose. With noticeable probability the second prover will use a strategy different than the first prover and the verifier will catch the mismatch. Upon executing the protocol many times the soundness error can be reduced until negligible. □

The second observation is that the provers may employ an optimal strategy to achieve perfect completeness. If the instance is indeed in the language as claimed, then regardless what questions the verifier asks, the provers are able to succeed in proving membership. Perfect completeness for MIPs is analogous to perfect completeness for IPs proved by [FGM$^+$89].

We mention a third additional result proved subsequently by a series of papers culminating in [FL92]. Analogous to how the IP hierarchy generated by the number of rounds collapses, so does the MIP hierarchy. But the MIP hierarchy collapses in a stronger sense than the IP hierarchy. In the case of IP any proof system with a constant number of rounds may be replaced by a proof system with a single round. In the case of MIP any proof system with a polynomial number of rounds may be replaced by a proof system with a single round.

Many other additional questions and results regarding multi-prover interactive proofs have been explored but they are beyond our scope. For example, it is natural to ask about public vs private coin MIPs, but there is no natural way to even define exactly what a public-coin proof would mean in the multi-prover setting. The verifier cannot reveal all its coins to both provers or else soundness is compromised. The verifier might not be able to partition its coins and send part to one prover and the rest to the other because a single coin may well determine questions to both provers. Somehow the verifier must reveal all of its randomness but sharing with each prover precisely the randomness used to determine questions for that prover. Another unanswered question is in regard to the parallel composition of MIPs.

Lastly, a natural question with a known answer is in regard to zero-knowledge, and indeed all of MIP can be proven in zero-knowledge [FK94].

2.3.3 MIP = NEXP

In [FRS88] it is proven that any language having a multi-prover interactive proof system must be computable in nondeterministic exponential time, that is $\mathcal{MIP} \subseteq \mathcal{NEXP}$. Soon following, [BFL91] proved that every language computable in nondeterministic exponential time has a multi-prover interactive proof system, that is $\mathcal{NEXP} \subseteq \mathcal{MIP}$. Therefore $\mathcal{MIP} = \mathcal{NEXP}$.

Due to the equivalence of MIP and NEXP it is worth thinking about the nature of NEXP. The complexity class NEXP can be thought of relative to NP which characterizes languages computable in nondeterministic polynomial time. Analogous to how we may interpret NP as the set of languages verifiable in polynomial time with polynomial size witnesses, so may we interpret NEXP as the set of languages verifiable in exponential time with exponential size witnesses. In fact, since our context is proof systems, these alternative definitions are more fitting than the traditional definitions. But we must remember that these proof system characterizations of NP and NEXP are not interactive except trivially in that the prover sends a static proof and the verifier verifies. The verifier never sends anything to the prover, which is essential in all types of interactive proof systems.

If we treat the complexity class P as the set of languages efficiently computable (rather than BPP), then noting that the containment $\mathcal{P} \subseteq \mathcal{NEXP}$ is known to be strict, this is the first time that languages not efficiently computable may be efficiently verified. In other words, languages not computable in polynomial time can be verified (probabilistically) in polynomial time. But since the verification is probabilistic, perhaps it is more fitting to allow the computation to be probabilistic, in which case we are left with the containment $\mathcal{BPP} \subseteq \mathcal{NEXP}$ which is not known to be strict. The holy grail of such computation vs verification asymmetry is the P vs NP problem.

2.3.4 The power of physical separation

A multi-prover interactive proof system has the same setup as a regular interactive proof system except there are multiple provers and they are forbidden from communicating while proving. The physical separation of the provers is an assumption that allows for even more powerful results than assuming a computationally bounded prover. These two types of assumptions, limited communication vs limited computation, are of very different character. It is interesting to think about how the traditional assumptions of limited computation underlying cryptography could be exchanged for assumptions of limited communication. We comment on this at the end.

A natural analogy between MIPs and real life is illustrated by the case of criminal suspects physically separated for interrogation. To quote the original paper [BOGKW88] of MIPs on this analogy,

> One may think of this as the process of checking the alibi of two suspects of a crime (who have worked long and hard to prepare a joint alibi), where the suspects are the provers and the verifier is the interrogator.

Indeed, just as two suspects may conspire before interrogation, so may the provers conspire before the proving process begins, that is before the verifier asks its first question. Since the provers are computationally unbounded, they may have a plan on how to answer every possible set of questions the verifier may pose them. But the verifier is still able to catch them if they lie because the verifier asks the provers related questions that must be answered in conjunction if the answer is to be a believable lie. Since the provers are separated they are unable to answer in conjunction and must choose their answers independently. We mention that neither the number of questions the verifier asks nor the order in which the verifier queries the provers must be secret and in fact such secrecy adds nothing. Therefore asynchronous communication models among the provers and verifier are unneeded and a synchronous communication model suffices. In the case of interrogation, analogously, the suspects are likely aware in which room the interrogator is currently located.

The type of cross-examination that takes place in MIPs is not exactly the kind that takes place in an interrogation. In an interrogation the suspects are often treated homogeneously, such as in the beginning when they may each be asked to tell their version of the story. Multi-prover proof systems, on the other hand, usually treat the provers inhomogeneously, such as one prover doing the majority of the work and a second prover answering questions to confirm the validity of the first prover's work. For example, to test a function f for linearity we may select two inputs α and β at random, ask the first prover for $f(\alpha)$ and $f(\beta)$, ask the second prover for $f(\alpha + \beta)$, and confirm the first prover's answers are correct by comparing via linearity with the second prover's answer. In this case the first prover does twice the work of the second prover.

The power of physical separation has been extended further using quantum computing. In particular, MIPs in the quantum setting are applicable to languages beyond NEXP. In the quantum setting the provers are all-powerful quantum computers that may share an entangled state but are otherwise prohibited from communicating. Exploration in this direction culminated in [JNV$^+$20], showing that the set of languages provable by such quantum provers is equal to the set of languages recognizable in a finite amount of time. An example is the undecidable halting problem because if a program ever stops it stops after a finite amount of time. Clearly physical separation is a powerful assumption.

Lastly we comment that the possibility of replacing intractability assumptions with physical separation assumptions in practice is exciting, but we currently do not know how to enforce physical separation. When two provers are aware of each other it seems hopeless to prevent them from communicating by some means. Another approach is to choose the provers such that they are not aware of each other. But in what case would two provers wish to prove the same statement and yet be unaware of each other? Even if at the beginning they were unaware of each other, they could use the identity of the statement being proven as a means to locate each other.

2.4 Algebraic approaches to proof systems

All proof systems used in practice today use algebraic techniques. In contrast, many of the early constructed proof systems and even many theoretically-oriented proofs systems constructed today, rely on combinatorial techniques instead. The first proof system that

appears to have employed algebraic techniques is [LFKN92], and the paper was appropriately titled 'Algebraic Methods for Interactive Proof Systems.'

The methods used in both the proof by [Sha92] that $\mathcal{IP} = \mathcal{PSPACE}$ and the proof by [BFL91] that $\mathcal{MIP} = \mathcal{NEXP}$ rely on the algebraic techniques developed in [LFKN92]. The purpose of [LFKN92] is constructing a proof system for the #P-complete problem of computing the permanent of a square binary matrix. We mention that every language in the polynomial-time hierarchy, that is the complexity class PH, is reducible to the class #P, and therefore the proof system of [LFKN92] suffices for all languages in PH, that is $\mathcal{PH} \subseteq \mathcal{IP}$.

The techniques of [LFKN92] involve representing the permanent via a multivariate polynomial. Using a polynomial to represent the permanent was a previous idea used in [Val79] to prove the average-case hardness of computing the permanent. It is worth mentioning that the permanent of a matrix is related to the determinant of a matrix, and both only make sense for square matrices. Yet the permanent is much harder to computer than the determinant, and as quoted by [Val79],

> We do not know of any pair of functions, other than the permanent and determinant, for which the explicit algebraic expressions are so similar, and yet the computational complexities are apparently so different. Leslie Valiant in [Val79]

The core contribution of [LFKN92], exploited by subsequent papers, is an implicit technique to probabilistically reduce verifying two evaluations of any low-degree polynomial to verifying only one evaluation. To see the idea explicitly for the special case of low-degree multivariate polynomials representing permanents, consult [LFKN92]Lemma 4. With this technique in hand, they apply it numerous times in a recursive fashion to achieve an efficient proof system for #P.

Subsequent papers extracted the implicit techniques of [LFKN92] and refined the ideas. The work [Sha92] modified the methods to work for quantified boolean formulas which characterize PSPACE, thus showing every language in PSPACE has an interactive proof. Independently, [BF91] also built upon the work of [LFKN92] to construct proof systems similar to that of [Sha92]. But the proof systems in [BF91] basically served as arithmetic-oriented approaches to proving membership in #P and some other languages, and didn't reach as far as PSPACE. Yet the authors soon followed up with another work [BFL91] to show every language in NEXP has a multi-prover interactive proof. The techniques employed by all these works came to be referred to as 'arithmetization' techniques because they used arithmetic for representations. Furthermore, they arithmetic theorems to prove properties of these representations.

If we are to glean the core contributions of the series of works in the preceding paragraph as far as proof systems are concerned (rather than complexity theory), we settle on two related protocols that we believe also serve as illustrations for the ideas contained in those works. The first protocol reduces verifying two evaluations of a low-degree polynomial to verifying one evaluation. The second protocol recursively invoked the first protocol to reduce verifying the sum of many evaluations of a low-degree polynomial to verifying a single evaluation. The second protocol has come to be known as the 'sumcheck' protocol, though this name was not used in any of the works above. These two techniques are described in `predicate_reduction/..` While we do not describe them here, we briefly describe how the sumcheck protocol can be used as a proof system for #P, in particular as a proof system

for counting the number of satisfying assignments to a boolean circuit. Suppose the circuit has n variables. We may arithmetize the circuit into a polynomial $f\colon \mathbb{Z}^n \to \mathbb{Z}$, leaving us to count the number of boolean inputs that yield output 1. That is, we must count the size of the set $|\{f(x)|x \in \{0,1\}^n\}|$, which is equal to $\sum_{x\in\{0,1\}^n} f(x)$. Using the sumcheck protocol we can reduce verifying this sum to verifying the evaluation of f at some random input $r \in \mathbb{Z}^n$.

2.5 Defining interactive proofs

The interaction of two machines, such as a prover P and a verifier V with their respective inputs, is given special notation. While the notation applies to any two interacting machines, we note it was developed for the purpose of an interacting prover and verifier. The purpose of the notation is to succinctly capture the outcome of the interaction in a way that can be inserted into probability equations and such. In fact, there are two outcomes of the interaction, those are the two outputs of the two interacting machines. The notation puts the two machines between angled brackets to represents the output of the machine on the right. In order to represent the output of the machine on the left, positions must be swapped. Thus the notation is asymmetric. For example, the output of the verifier after interaction is denoted $\langle P(\cdot), V(\cdot)\rangle$, and a proper verifier V should presumably always output a boolean answer indicating acceptance or rejection. The output of the prover after interaction is denoted $\langle V(\cdot), P(\cdot)\rangle$, but the prover's output is seldom considered. A formal definition of interacting machines in terms of Turing machines that communicate using various tapes can be found elsewhere such as in the original zero-knowledge paper [GMR85].

We now formally define interactive proof systems.

Definition 1 (Interactive proof systems)**.** *An* interactive proof system *for a set S is a dynamic proof system wherein the prover and verifier interact exchanging messages. The prover algorithm P is unrestricted with unbounded computational resources. The verifier algorithm V is restricted to probabilistic polynomial time. Such an asymmetry of computational resources focuses analysis on the verification process rather than the proving process.*

In addition to the common input $x \in S$, we assume both the prover and verifier have private auxiliary inputs denoted y and z respectively. We may suppose all inputs are described in bit-strings. All metrics such as the lengths of the auxiliary inputs and the running time of the verifier are measured in terms of the length of the instance $|x|$. The verifier V has two possible outputs, one to indicate acceptance and one to indicate rejection. Below we use the bit 1 to indicate acceptance and the bit 0 to indicate rejection.

The following two conditions must be met. The first condition captures what should occur if the prover and verifier both behave. The second condition captures what should occur if the prover misbehaves and the verifier behaves. What should occur if the prover behaves and the verifier misbehaves is captured by additional conditions such as zero-knowledge, defined separately, see section 2.6.

- ***Completeness:***

  The completeness condition ensures that in the use of both the prescribed prover P and verifier V strategies the proving process will succeed in that V will accept the proof with

high probability.

For every $x \in S$, the interaction of P and V on common input x results in rejection by V with probability bounded below one half, such as probability at most $1/3$.

$$\Pr(\langle P(y), V(z)\rangle(x) = 0) \leq 1/3 \tag{2.7}$$

We may then infer a lower bound on the probability of acceptance.

$$\Pr(\langle P(y), V(z)\rangle(x) = 1) > 2/3 \tag{2.8}$$

- ***Soundness:***

  The soundness condition ensures that in the use of any prover strategy P^* trying to prove a false statement, the proving process will fail in that V will reject the proof with high probability.

  For every $x \notin S$ and every prover P^*, the interaction between $P^*(x)$ and $V(x)$ on x results in rejection by V with probability bounded above zero, such as probability at least $1/3$.

$$\Pr(\langle P^*(y), V(z)\rangle(x) = 0) \geq 1/3 \tag{2.9}$$

  We may then infer an upper bound on the probability of acceptance.

$$\Pr(\langle P^*(y), V(z)\rangle(x) = 1) < 2/3 \tag{2.10}$$

While the probability of completeness and soundness error may seem large, interactive proofs can be repeated multiple times such that the error becomes negligible.

2.6 Defining zero-knowledge

Here we reduce definitions of zero-knowledge to definitions of ensemble indistinguishability. There are three such definitions, namely perfect, statistical, and computational indistinguishability. To see the relevant definitions of ensemble indistinguishability consult appendix A. The three types of indistinguishability gives rise to three corresponding types of zero-knowledge, namely perfect, statistical, and computational zero-knowledge respectively. When zero-knowledge is reference without a type it usually refers to computational zero-knowledge but it is the most encompassing definition of zero-knowledge.

The idea behind the definition of zero-knowledge is if one can simulate the interaction without any help from the prover then one certainly does not gain knowledge by interacting with the prover. We first present definitions of zero-knowledge based on the simulation of the output of the verifier. Second we present definitions based on the simulation of the messages received by the verifier. Then we present relaxed alternatives that may apply to either kind of definition, and lastly we mention the complexity classes inspired by zero-knowledge definitions.

Often one may better understand a definition by examining how many alternating quantifiers are involved in the definition and what purposes they each serve. In the case of

zero-knowledge definitions, there are usually three quantifiers. The first is universal and quantifies over verifiers. The second is existential and quantifies over simulators. The third is universal and quantifies over instances of the language, that is the language in which the prover wishes to prove membership. In the special case of honest-verifier zero-knowledge defined below, the first quantifier is dropped, and indeed this yields a weaker definition.

2.6.1 Simplified and auxiliary-input ZK

Recall from the interaction notation of definition 1 that $\langle A, B\rangle(x)$ denotes the output of machine B after interacting with machine A on common input x.

Definition 2 (Simplified zero-knowledge)**.** *An interactive proof system (P, V) for a language L is **zero-knowledge without auxiliary input** if for every p.p.t.verifier V^* there exists a p.p.t.simulator M^* such that the following two ensembles are indistinguishable.*

$$\{\langle P, V^*\rangle(x)\}_{x\in L} \text{ and } \{M^*(x)\}_{x\in L} \tag{2.11}$$

The above definition was the original definition of zero-knowledge, that is the one proposed in [GMR85]. Subsequently, [GO90] proposed an augmented definition called 'auxiliary-input zero-knowledge', allowing the verifier to have any auxiliary input before beginning interaction with the prover. The auxiliary input is intended to account for information the verifier may have already received from the prover in prior interactions should the protocol be executed as a subprotocol inside a larger protocol. For example, many of the early proof systems used cut-and-choose technique in which the prover would cut the witness into parts and the verifier would choose to see one part. These have a high soundness error and therefore in order to attain negligible soundness error the protocol must be executed many times. Indeed, [GO90] show that auxiliary-input zero-knowledge proof systems are closed under sequential composition, meaning the sequential execution of auxiliary-input zero-knowledge protocols is itself an auxiliary-input zero-knowledge protocol.

Definition 3 (Auxiliary-input zero-knowledge)**.** *An interactive proof system (P, V) for a language L is auxiliary-input zero-knowledge if for every p.p.t.verifier V^* there exists a p.p.t.simulator M^* such that the following two ensembles are indistinguishable.*

$$\{\langle P, V^*(y)\rangle(x)\}_{x\in L, y\in\{0,1\}^{\text{poly}|x|}} \text{ and } \{M^*(x, y)\}_{x\in L, y\in\{0,1\}^{\text{poly}|x|}} \tag{2.12}$$

The y values are the auxiliary inputs. The running times of V^* and M^* are measured in terms of their input lengths, both of which are $|x| + |y|$.

Remark 1 (Irony of theory vs practice)**.** *Note that in auxiliary-input zero-knowledge the verifier but not the prover has access to an auxiliary input. Somewhat ironically, the opposite occurs in the definition of zero-knowledge we use in practice. Specifically, the prover but not the verifier has access to an auxiliary input, in particular the prover has access to the NP witness. This is an example where theoretical and practical motivations (such as those of [GO90] and ours respectively) lead in different directions.*

In theory, the prover has unbounded computational resources and therefore does not need an auxiliary input. In practice, the prover runs in polynomial time and may need the witness

as an auxiliary input. In theory, P may leak information about the witness in such a way that V may aggregate the information across multiple executions of the protocol and then convert the information into knowledge. The verifier is given auxiliary input to represent this information, and the definition of auxiliary-input zero-knowledge ensures that even with this information zero-knowledge still holds. In practice, P does not leak any information about the witness and the verifier has no secret information to hide from the prover, so any verifier auxiliary input may be considered part of the common input.

Most known zero-knowledge proof systems are naturally auxiliary-input zero-knowledge. But [GK96] constructed a proof system satisfying simplified zero-knowledge but not auxiliary-input zero-knowledge. Since most proof systems already satisfy auxiliary-input zero-knowledge and it is necessary for sequential composition, auxiliary-input zero-knowledge is a natural default definition for zero-knowledge.

2.6.2 View-based and honest-verifier ZK

We now proceed to discuss several more variations of zero-knowledge. Each of them may be either simplified zero-knowledge or auxiliary-input zero-knowledge. For completeness, we state the definitions as auxiliary-input zero-knowledge, but the auxiliary inputs could just as well be omitted to obtain simplified versions. Like before, we assume the running times of the verifier and simulator are measured in the total lengths of their inputs.

First we present a definition of zero-knowledge from [GO90] based on what the verifier receives from the prover, rather than what the verifier outputs after interacting with the prover. We may refer to this as 'view-based' zero-knowledge, whereas we may refer to the previous definitions as 'output-based' zero-knowledge.

Definition 4 (View-based zero-knowledge)**.** *For an interactive proof system (P, V), denote by $view(P, V^*(y), x)$ the **view** of the interaction as seen by V^*, consisting of the common input x, auxiliary input y, the random coins of V^*, and all messages sent from P to V^* throughout the interaction.*

We say (P, V) for language L is view-based zero-knowledge if for every p.p.t.verifier V^* there exists a p.p.t.simulator M^* such that the following two ensembles are indistinguishable.

$$\{view(P, V^*(y), x)\}_{x \in L, y \in \{0,1\}^{\text{poly}|x|}} \text{ and } \{M^*(x, y)\}_{x \in L, y \in \{0,1\}^{\text{poly}|x|}} \tag{2.13}$$

Observe how the view-based definition is a simple modification of auxiliary-input zero-knowledge. We have simply replaced $\langle P, V^*(y)\rangle(x)$ with $\text{view}(P, V^*(y), x)$. We now show the previous view-based definition is equivalent to the output-based definition.

Proposition 2 (View-based ZK is equivalent to regular ZK)**.** *A proof system is view-based zero-knowledge if and only if it is output-based zero-knowledge.*

Proof. To prove the forward direction, note that given the view of a verifier interacting with a prover one may compute the output of the verifier. This is because the view not only includes the messages from the prover but also the inputs including the random input and any auxiliary input, such that the output is a polynomial-time deterministic function of the

view. Therefore if the view contains no knowledge then the output contains no knowledge, so view-based zero-knowledge implies output-based zero-knowledge.

Towards proving the backward direction, first note that even if a protocol is output-based zero-knowledge with respect to a certain verifier, we can't say it is view-based zero-knowledge with respect to that verifier because the view may contain knowledge while the output does not. This trouble can only be circumvented because the definition of zero-knowledge universally quantifies over all verifiers. In order to prove the view of a verifier V^* contains no knowledge we prove the output of a related verifier $V^{*'}$ contains no knowledge. The verifier $V^{*'}$ uses V^* as a black-box and $V^{*'}$ basically delegates its own role as a verifier to V^* by passing all inputs to V^* and acting as an intermediary between V^* and the prover. While acting as an intermediary, $V^{*'}$ records all messages sent to V^*. By the end, $V^{*'}$ has captured the entire view of V^*. When V^* sends its final output, $V^{*'}$ does not output the same, but rather discards the output of V^* and instead outputs the view of V^*. So basically this shows that the set of views of all verifier is a subset of the set of outputs of all verifiers. Therefore if the set of outputs contain no knowledge then the set of views contain no knowledge, so output-based zero-knowledge implies view-based zero-knowledge. □

Now we move to what's called 'honest-verifier' zero-knowledge which is formulated in terms of view-based zero-knowledge. A common way to relax the requirements of zero-knowledge it to only require security against the single prescribed verifier strategy rather than against all possible verifier strategies. A verifier that uses the prescribed verifier strategy is called an 'honest verifier.' An honest verifier will not ask maliciously intended questions to the prover, but may still examine the responses from the prover and try to extract knowledge from them. The output of the prescribed prover is only a single bit indicating acceptance or rejection. While the single output bit of the verifier may not contain knowledge, the view of the verifier may well contain knowledge. Therefore in order to prove zero-knowledge with respect to an honest-verifier, we must use view-based rather than output-based zero-knowledge.

Definition 5 (Honest verifier zero-knowledge)**.** *A proof system (P, V) for a language L is* ***honest verifier zero-knowledge*** *if there exists a p.p.t.simulator M such that the following two ensembles are indistinguishable.*

$$\{view(P, V(y), x)\}_{x \in L, y \in \{0,1\}^{\text{poly}|x|}} \text{ and } \{M(x, y)\}_{x \in L, y \in \{0,1\}^{\text{poly}|x|}} \tag{2.14}$$

The work [GSV98] proves that honest-verifier statistical zero-knowledge implies statistical zero-knowledge in general. For the special case of public-coin honest-verifier zero-knowledge, the transformation also applies to computational zero-knowledge rather than just statistical and perfect zero-knowledge. Notably, the transformation of [GSV98] does not rely on intractability assumptions.

While honest-verifier zero-knowledge is insufficient in general because there's no guarantee the verifier will use the prescribed strategy, it is often sufficient in practical proof systems. Most proof systems used in practice employ the Fiat-Shamir transformation to convert the proof to a non-interactive proof, effectively leaving the verifier strategy up the prover. The prover should naturally choose the prescribed verifier strategy if it wishes to prove the statement while maintaining privacy. The reason the notion of honest-verifier zero-knowledge is important to us is precisely this non-interactive scenario popular in practice.

2.6.3 Relaxations of zero-knowledge

First we present a relaxation called the 'liberal' notion of zero-knowledge [Gol01]p.205. This relaxation applies to all three forms of zero-knowledge, that is perfect, statistical, and computational zero-knowledge.

Definition 6 (Perfect zero-knowledge)**.** *A proof system (P, V) for some language L is **perfect zero-knowledge in the liberal sense** if for every p.p.t.simulator V^* there exists a probabilistic, expected polynomial-time simulator M^* such that the following two distributions are identical.*

$$\{\langle P, V^*(y)\rangle(x)\}_{x\in L, y\in\{0,1\}^{\text{poly}|x|}} \text{ and } \{M^*(x,y)\}_{x\in L, y\in\{0,1\}^{\text{poly}|x|}} \tag{2.15}$$

The expectation of the simulator's running time is taken over its internal coin tosses.

In the above definition lets the simulator run in expected polynomial-time but never fail to output an identical distribution. An alternative lets the simulator run in strict polynomial time but fail with negligible probability.

Definition 7 (Relaxed perfect zero-knowledge [Gol01, p. 202])**.** *A proof system (P, V) for some language L is **perfect zero-knowledge with bounded error** if for every p.p.t.verifier V^* there exists a p.p.t.simulator M^* such that the following two conditions hold.*

- *With probability bounded away from one the simulator fails by outputting the symbol $\perp$. For the constant bound $1/2$ this means for every $x \in L$ and $y \in \{0,1\}^{\text{poly}}|x|$ we have*

$$\Pr M^*(x,y) = \perp \leq \frac{1}{2} \tag{2.16}$$

  The probability is taken over x and y and the internal coin tosses of M^*. One may invoke the simulator enough times to render a negligible probability of error. Or one could strengthen the definition and require the initial probability of error to be negligible.

- *Consider the distribution of random variable M^* conditioned on it not outputting $\perp$. Let m^* be the random variable describing this conditional distribution. The following two distributions must be identical.*

$$\{\langle P, V^*(y)\rangle(x)\}_{x\in L, y\in\{0,1\}^{\text{poly}|x|}} \text{ and } \{m^*(x,y)\}_{x\in L, y\in\{0,1\}^{\text{poly}|x|}} \tag{2.17}$$

Remark 2 (The symbol $\perp$)**.** *The symbol $\perp$ is an upside down 'T', as if signifying something antithetical to 'true.' It denotes that something went wrong, as in the above case when the simulator fails. Another typical use of the symbol is the output of a verifier to indicate rejection.*

The latter bounded-error alternative implies the former expected polynomial-time alternative, but not vice versa. The implication follows by repeatedly executing the bounded-error simulator until it succeeds. The difference is meaningful because there exist proof systems zero-knowledge with respect to the latter but not the former relaxation [Gol01]p.205. Yet

the latter alternative with strict polynomial time is preferred over the former alternative with expected polynomial-time because expected-polynomial time is not well understood compared to strict polynomial-time. For example, expected polynomial-time is not closed under composition. Regarding the subtleties of expected polynomial-time see [Gol11].

Unlike the former relaxation, the latter relaxation is only meaningful for perfect zero-knowledge and has no affect in the context of statistical or computational zero-knowledge. If two distributions are statistically or computationally close, they remain so after a negligible error is applied to one of them, thus statistical and computational zero-knowledge are preserved when the simulator errs with negligible probability.

2.6.4 Zero-knowledge complexity classes

For every definition of zero-knowledge one may formulate the complexity class of all languages in which membership may be proved under that definition of zero-knowledge.

Definition 8 (ZK complexity classes)**.** *The classes of languages with perfect, statistical, and computational zero-knowledge proof systems are denoted by $\mathcal{PZK}$, $\mathcal{SZK}$, and $\mathcal{CZK}$ respectively. The generic notation $\mathcal{CZK}$ is taken to be an alias for $\mathcal{CZK}$ because computational zero-knowledge is the largest class.*

The classes of languages with honest-verifier perfect, statistical, and computational zero-knowledge proof systems are denoted by $\mathcal{HPZK}$, $\mathcal{HSZK}$, and $\mathcal{HCZK}$ respectively. The class $\mathcal{HZK}$ is an alias for $\mathcal{HCZK}$.

There is no particular notation for the classes of languages having relaxed forms of zero-knowledge proof systems.

Proposition 3 (Inclusion chain of ZK classes)**.**

$$\mathcal{BPP} \in \mathcal{PZK} \in \mathcal{SZK} \in \mathcal{CZK} \in \mathcal{IP} \tag{2.18}$$

Proof. The first inclusion $\mathcal{BPP} \in \mathcal{PZK}$ holds because $\mathcal{BPP}$ naturally has a perfect zero-knowledge proof system, namely that in which the prover is non-existent and the verifier attempts to compute the answer on its own with bounded soundness error. The inclusions $\mathcal{PZK} \in \mathcal{SZK}$ and $\mathcal{SZK} \in \mathcal{CZK}$ clearly hold by the definitions of perfect, statistical, and computational indistinguishability of distribution ensembles. The last inclusion $\mathcal{CZK} \in \mathcal{IP}$ because any non-trivial zero-knowledge proof system is an instance of an interactive proof system. □

2.7 Alternatives to zero-knowledge

We mention two alternative notions to zero-knowledge that have been proposed in the literature. Upon discovering more we may summarize them as well. The first proposed is called 'minimum disclosure proofs' introduced by [BCC87], and the second is 'witness hiding proofs' introduced by [FS90]. Both notions are weaker than zero-knowledge and while they both have their benefits, zero-knowledge remains the standard notion for prover privacy.

2.7.1 Minimum disclosure proofs

In the minimum disclosure model of [BCC87], the prover discloses nothing about the witness to the verifier. They call it 'minimum' because the prover must still disclose whether or not the statement is true. On the other side of the spectrum there's maximum disclosure proofs revealing the entire witness. For example, the prover simply handing the verifier the witness constitutes a valid maximum disclosure proof. It may seem that minimum disclosure meets the needs for privacy just as well as zero-knowledge, but for a subtle reason this may not be so. Minimum disclosure is a weaker notion than zero-knowledge because the prover may still reveal knowledge to verifier, just as long as this knowledge does not concern the witness. For example, the verifier may gain knowledge about a commitment scheme used throughout the proof by strategically querying the all-knowledgeable prover. More generally, it may not be known how to simulate the interaction with the prover, and the verifier may have gained knowledge through the interaction even when unclear what knowledge was gained.

One may ask why zero-knowledge is the popular choice for privacy when minimum disclosure also ensures witness privacy. In fact, not only does minimum disclosure suffice for privacy, but it appears to be the tightest definition of privacy in that any proof system hiding the witness satisfies minimum disclosure, yet many such proof systems may not be fully simulatable and thus not satisfy zero-knowledge. Nevertheless, zero-knowledge remains the standard. Perhaps some proof systems wish to ensure the verifier gains nothing from interaction, even if irrelevant from the witness. More likely, however, is that the simulation-based definition of zero-knowledge guarantees zero knowledge is leaked, whereas the definition of minimum disclosure leaves one to determine exactly what knowledge might be leaked and whether or not such leakage is a problem. In fact, [BCC87] do not even provide a formal definition of minimum disclosure. Or perhaps the reason is simply that zero-knowledge came first, and while overly cautious the definition has been successful enough that there is no need to seek a tighter definition like minimum disclosure.

2.7.2 Witness hiding proofs

Before introducing the notion of witness hiding, [FS90] introduces the notion of witness indistinguishability. A proof system is witness indistinguishable when the verifier cannot distinguish which witness was used by the prover. Specifically, a proof system is witness indistinguishable if for any large enough instance, any verifier auxiliary input, and any two witnesses, the two views of the verifier (sequences of messages sent by the prover) corresponding to the two witnesses are indistinguishable. As with zero-knowledge, one may consider perfect, statistical, or computational indistinguishability. Witness indistinguishability is only nontrivial when there are multiple witnesses, which is often the case.

A proof system is witness hiding when the verifier cannot learn the witness from the interaction. They prove that if a proof system is nontrivially witness indistinguishable then it is also witness hiding. It may seem that witness hiding meets the needs for privacy just as well as zero-knowledge, but for a subtle reason this may not be so. Witness hiding is a weaker notion than zero-knowledge because the verifier may learn part of the witness, just as long as the verifier doesn't learn the entire witness. For example, in the case of a one-way permutation the witness is unique and witness hiding guarantees nothing in this

case. In fact, witness hiding is even weaker than minimum disclosure because any minimum disclosure implies witness hiding by the trivial fact that if the verifier learns nothing of the witness then the verifier certainly doesn't learn the whole witness.

The core benefit of witness hiding proof systems, and probably the motivation behind them, is that they may be composed arbitrarily with other protocols without sacrificing security. This is not so with zero-knowledge. In particular, parallel composition is appropriate for witness hiding protocols but not for zero-knowledge protocols in general. An example of zero-knowledge failing in the setting of parallel composition is shown by [FS90] as motivation for witness hiding. They construct a simple zero-knowledge proof system that when executed twice in parallel allows the verifier to use the messages of one prover as queries to the other prover in a way that extracts the entire witness.

A notable use of witness indistinguishability beyond implying witness hiding is it's use in [FLS90] for constructing zero-knowledge noninteractive proof systems that support a polynomial number of proofs with the same setup. Such are most proof systems used in practice, including the one's developed in this project, but they do not use the same technique as [FLS90] or witness indistinguishability in general.

Chapter 3

The Saga Batching Model

3.1 Motivation

In this work we propose a new scenario of practical interest in the real world, and we provide two solutions for such a scenario. The problem our solutions address is the real world problem in which a prover wishes to prover many proofs of the same type of statements to a verifier, at minimal cost for both the prover and verifier. Schemes providing a solution for this basic scenario are called batching schemes because they batch together many proofs. We are concerned, however, with batching schemes satisfying a certain kind of efficiency, space complexity, as we now motivate. Proof systems known today have at least one of two problems for use in practice, particularly for the purpose of batching many proofs together.

The first problem is that many proof schemes provide large proof sizes far to large for the verifier to be appropriate for the solution. In fact, many proof systems send proofs so large the verifier is required to perform more computation than it would if the prover had just sent an NP-witness. The benefit of these proofs, however, is that they satisfy zero-knowledge with minimal complexity assumptions, thus making the large proof size worth it. An example is the recently proven breakthrough result of single round interactive proofs only assuming the hardness of the lattice learning with errors problem [PS19], which follows a line of work consisting of [CLW19], [CCH$^+$19], and [CCH$^+$18]. When faced with the task of batching many proofs together, while the prover may be able to operate efficiently, the proof size is simply far to large for the verifier's available space complexity.

The second problem is that when proof sizes are small, the prover has large overhead in computing the proof. In particular, the proving time for a single proof is super-linear, usually by a logarithmic or poly-logarithmic factor, in the size of the witness. The additional factor arises often from the need to compute a large polynomial division or a large polynomial interpolation. A line of such proof systems of prime interest today is [BSBHR17], [BSBHR18], [BSCR$^+$18], and further citations therein. The study of [WSR$^+$15] shows these types of proofs consume expensive amounts of space for proving, more space than every day electronic devices possess. When faced with the task of batching many proofs together, while the proof size may be small, the prover's overhead in proving all of these proofs in a single proof is far to large for the prover's space complexity.

We avoid both of these problems by having the prover send many small proofs over a

period of time, rather than having the prover send one big proof at once. This way the prover only needs the space complexity for proving one small proof at a time, and the verifier only needs the space complexity for verifying one proof at a time. This scenario may seem naive in that it may seem equivalent to the prover just proving each proof consecutively using some existing proof system. But via our novel methods we provide two constructions that are more efficient than this naive solution. In particular, we allow the prover and verifier to amortize the proofs, such that the overall cost is minimal.

Three metrics are still worth measuring to judge the quality of such a solution. The first is the cost of setting up the protocol, specifically the amount of data that must be exchanged prior to the first proof. The second is the proof size on a per proof basis. The third is the cost of wrapping up the protocol, specifically the amount of data that must be exchanged after the last proof. Only after this is the verifier expected to make a decision on whether to accept or reject all proofs in the batch.

We call such a solution a 'saga batching scheme' because of how it takes place over a long period of time. Every time the prover sends a proof, its like the prover telling the verifier part of a long story. Such longer, ever ongoing stories are called sagas. At the end, when the protocol is wrapped up and verifier is finally able to decide whether to accept or reject the proofs, is like the pinnacle of the saga story.

3.2 Related work

We are not the first to propose schemes for amortizing proofs. In fact, nearly all proof systems are inherently a form of amortizing verifications. But such amortization is of the one-shot type, where it can only be done once. We seek, on the other hand, to amortize an arbitrary number of proofs keeping the space complexities of the prover and verifier constant. The only known technique for achieving such arbitrary compression of proofs is via recursive proof composition. Recursive proof composition, however, is highly non-trivial and presents many challenges both theoretical and practical in flavor. Several works exist in the area, however, and they share the closest goals to ours. We mention them now.

First we clarify what recursive proof composition is exactly. When a verifier verifies a proof, the verifier is performing a computation, so it is natural to ask whether the task of verifying a proof can itself be proven via a proof. This way a verifier ends up verifying proofs of proofs of proofs, etc, compressing an arbitrary number of proofs. The first work in the direction of recursive proof composition was [Val08], introducing the notion of 'incrementally verifiable computation' in which there is a sequence of proofs such that each proves the validity of the previous proof and one additional statement. While this first work was theoretical in nature, the second [BSCTV14a] was practical in nature, though not practical enough to use in practice. It utilized proof systems based on pairings cryptography due to their tiny proof size, and recursed on them by using cycles of elliptic curves such that pairings on one curve could verify pairings on the other. More recently, [COS19] achieved a more practical method for recursive proof composition using the line of work mentioned previously with large prover overhead.

All proof systems up to this point had achieved recursive composition directly by one proof performing a full verification of a previous proof. Independently from our work which

realized the same insight, the recent work of [BGH20] changed this. Instead of a proof fully verifying another, the proof may instead amortize the verification of other proofs. While the work of [BGH20] was rather informal, a subsequent work [BCMS20] formalized the insight calling such amortization schemes 'accumulation schemes.' They admit, however, that they are unaware of any post-quantum secure accumulation scheme as they only provide ones with respect to discrete logs and RSA.

Now we describe how our work compares to these existing works. Like [BGH20] and [BCMS20], we achieve our results with respect to an arbitrary number of proofs by using amortization rather than direct recursive proof composition. Our construction using discrete logs is quite like an accumulation scheme as defined in [BCMS20]. But it is a novel form of amortization perhaps of interest in its own value. One may compare the format of the proof to that of [BSCR$^+$18], which is not an amortization scheme but uses a similar computational model for proving statements. The technique of [BSCR$^+$18] for this computational model requires the prover to make a commitment not just to the original witness, but also to several derivatives of the witness which are actually all larger than the witness itself. This is costly. Our technique of amortization completely removes the need for such redundant commitments.

Now regarding how our construction using lattices compares to others, we use a novel recursive amortization technique. While we use it for proving single proofs at a time, it could be used across proofs to form a post-quantum secure variant of an accumulation scheme as described in [BCMS20]. To our knowledge our construction is the first that is capable of achieving an arbitrary amount of amortization with post-quantum security. Our lattice scheme is not the first proof system using lattices, and in fact uses the same proof of knowledge as the work [BBC$^+$18]. Regarding our technique of recursion, recursive techniques are certainly not new to the ecosystem. In [BCC$^+$16], a recursive technique was proposed for discrete logs that was the basis for numerous subsequent works, such as the popular types of proofs called 'bulletproofs' presented in [BBB$^+$18], also in the discrete log setting. Regarding recursion in the lattice setting, ours is the first along with the very recent independent work [BLNS20], by one of the co-authors of the previous two works featuring recursion using discrete logs. Comparing the work [BLNS20] to ours, we may say ours works more efficiently in practice because [BLNS20] involves very large parameters, impractical at the moment for applications. Both of our works, however, due to utilizing lattices, are less practical than discrete log based constructions.

Lastly, we describe in what contexts saga batching schemes may be applicable in practice, justifying their importance as a solution. Proof systems are an exciting prospect for the blockchain ecosystem, but they require significant computational resources. They will only receive mass adoption when they can be deployed across mobile devices and implemented in smart contracts across blockchains. Both mobile devices and smart contracts, however, have minimal storage capacity and if they are to handle proof systems, whether for proving or for verifying, the proof system must require minimal space complexity. That is why we focus on space complexity. Time complexity is less of an issue in both theory and practice. In theory, time complexity is upper bounded by space complexity. In practice, mobile devices and smart contracts may do idle computations for a long period of time using minimal space. Indeed, the majority of battery usage in mobile devices as well as the majority of smart contract fees are incurred from giant fluctuations in computational intensity.

3.3 Saga batching model

Below we define what properties a saga batching scheme is expected to satisfy. Then we construct a model of computation for language recognition that we will use in both of our constructions of saga batching schemes.

Definition 9 (Saga batching scheme)**.** *A* ***saga batching scheme*** *is a public coin, interactive proof systems for batching many proofs together for the same type of statements. Any saga batching scheme must satisfy the following properties.*

Bounded prover: *The prover may be expected to have bounded computational resources. In particular, the prover may be said to be probabilistic polynomial time machine.*

NP language recognition: *The scheme can handle any language in the NP complexity class, such that the prover can prove any statement in the language. By definition, this is the complexity class consisting of languages that are efficiently verifiable. The purpose of the prover then is to make this verification process far more efficient.*

Completeness: *If the prover is honest, the verifier will accept all proofs in the batch with probability negligibly close to* 1*.*

Soundness: *If the prover is dishonest, the verifier will reject all proofs in the batch with probability negligibly close to* 0*.*

Space complexity: *The prover and verifier's space complexity is constant in the number of proofs proven. That is, regardless how many proofs are proven, both the prover and verifier maintain the same space complexity across the proofs.*

We use the following parameters in an effort to capture this notion.

1. Initial communication size*. This is the initial amount of data that must be exchanged between the prover and verifier before any proofs are proven.*
2. Per proof size*. This is the amount of communication sent from the prover to the verifier throughout the execution of a single proof.*
3. Final communication size*. This is the amount of data that must be exchanged between the prover and verifier after all proofs are executed, before the verifier can make its final decision.*

In order for both our constructions to satisfy the requirement of NP language recognition, we construct a language model now that suffices. Both of our constructions will be suited for proving membership in respective variants of this language, see definition 12 for the discrete log variant and definition 20 for the lattice variant. We call these programs arithmetic programs, and as we discuss after the definition, they are derived from equivalent models of computations presented in other works.

Definition 10 (Arithmetic programs)**.** *An* ***arithmetic program*** $(\mathbb{F}, \texttt{params}, A, B, C)$ *is a 5-tuple consisting of the following parts.*

- *The first component* $\mathbb{F}$ *is a finite field.*
- *The second component is a set of parameters which we now define.*
  1. $\texttt{constraintCount} \in \mathbb{N}$ *which we abbreviate* $\texttt{Ccount}$*.*
  2. $\texttt{statementSize} \in \mathbb{N}$ *which we abbreviate* $\texttt{Ssize}$*.*
  3. $\texttt{witnessSize} \in \mathbb{N}$ *which we abbreviate* $\texttt{Wsize}$*.*
  4. $\texttt{variableSize} \in \mathbb{N}$ *which we abbreviate* $\texttt{Vsize}$*.*

  The last parameter is formulated entirely dependent on the former parameters as follows.
  $$\texttt{variableSize} \coloneqq 1 + \texttt{statementSize} + \texttt{witnessSize}$$
- *The last three components are matrices* $A, B, C \in \mathbb{F}^{\texttt{Ccount} \times \texttt{Vsize}}$*. They express the logic of the program.*

Given an arithmetic program the goal is find statements that are correct with respect to the program, just like one may state that a certain input-output pair is correct with respect to some function in a codebase. In order to prove a statement valid, there exists the notion of a 'witness'. Intuitively, a statement is like an input and an output of the program that together are purportedly correct with respect to the program. The witness (like a witness in court) is a justification for the proposition that the input and output do indeed correspond to a correct execution of the program. A statement-witness pair may not necessarily be correct however. Either the statement may be false because there exists no witness, or the statement may be true but the witness does not correctly justify the statement. We use the notion of a witness below to define what it means for a statement-witness pair to satisfy a given arithmetic program.

Definition 11 (Arithmetic program satisfaction)**.** *A statement witness pair* $(s, w) \in \mathbb{F}^{\texttt{Ssize}+\texttt{Wsize}}$ ***satisfies*** *an arithmetic program* $(\mathbb{F}, \texttt{params}, A, B, C)$ *if the following conditions hold.*

- *The statement* s *is of size* $\texttt{statementSize}$ *and the witness* w *is of* $\texttt{witnessSize}$*.*
- *Let the variable vector* v *corresponding to statement-witness pair* (s, w) *be defined as* $v = (1, s, w) \in \mathbb{F}^{\texttt{Vsize}}$ *Then we require that the following equation holds in which '*$\circ$*' is the Hadamard product.*
  $$(Av) \circ (Bv) + (Cv) = 0$$
  To clarify, we may equivalently write
  $$\forall i \in [\texttt{Ccount}],\ \Big(\sum_{j=1}^{\texttt{Vsize}} A_{i,j} \cdot v_j\Big)\Big(\sum_{j=1}^{\texttt{Vsize}} B_{i,j} \cdot v_j\Big) + \Big(\sum_{j=1}^{\texttt{Vsize}} C_{i,j} \cdot v_j\Big) = 0$$

A language equivalent to the above in the proof setting was first introduced in [GGPR12]. In this work they were called 'quadratic arithmetic programs', abbreviated QAPs. The language was an extension to the arithmetic setting from the boolean setting of what the work called 'quadratic span programs,' abbreviated QSPs. Quadratic span programs in turn were an augmentation to the quadratic setting of a previously proposed model of computation, not capable of recognizing NP, called 'span programs' [KW93]. The idea behind the name is that the constraints span a certain space across all witnesses, and the goal is to find a witness in this span thus satisfying the constraints. In span programs the space spanned in linear, thus their lack of NP language recognition, whereas in quadratic span programs the space spanned in non-linear. Perhaps due to the success of the novel cryptographic scheme employed by [GGPR12] and its extensive deployment in the real world today such as in ZCash [BSCG$^+$14], the language model has become popular.

A subsequent work that redefined the language for its own purposes is [BSCTV14b], at which point it was given the name 'rank one constraint satisfaction', abbreviated R1CS. The idea behind the name is that the constraints that must be satisfied are of rank 1 in the multiplicative dimension due to the quadratic structure.

Theorem 1 (Arithmetic programs capture NP)**.** *Any NP language can be recognized by an arithmetic program.*

Proof. Our model of arithmetic programs is equivalent to that presented in [BSCTV14b]. See [BSCTV14b] section 2.3, for a proof that such programs are capable of simulating any arithmetic circuit, and thus recognizing any language in NP. □

3.4 Building blocks

Below we present three primary building blocks in our constructions. The first is the Schwartz-Zippel lemma, which can confidently be said to be the underlying tool used in all proof systems used in practice, and even most of those used in theory. From this lemma it immediately follows that if two low-degree polynomials are different, regardless their variacy, they will evaluate differently on almost all inputs. This provides a means for error detection, in which one randomly evalutes two polynomials to see if they are the same. Indeed, error detection can be viewed as the core of proof systems in that the verifier is tasked with detecting errors in a proof submitted by a prover.

The second building block is used extensively in both of our constructions. It is designed for the purpose of amortizing polynomial evaluations. Indeed, we call the protocol the **EvaluationReduction** protocol because it reduces the task of evaluating a polynomial at many points to evaluting it at a single point.

The third building block follows immediately from a recursive application of the **EvaluationReduction** protocol. It is called the 'sumcheck' protocol, and it is the fundamental building block of all proof systems based on multivariate polynomials.

At the end we introduce an important notational convention that will be employed constantly in both of our constructions.

3.4.1 The Schwartz-Zippel lemma

The following is a variant of the classic Schwartz-Zippel Lemma, which was intially proven independently first by [Zip79] and then by [Sch80].

Theorem 2. *Let f be a non-zero, v-variate polynomial over a field $\mathbb{F}$ of total degree at most t.*

$$\Pr_{x \in \mathbb{F}^v}\big[f(x_1, \ldots, x_v) = 0\big] \leq \frac{t}{|\mathbb{F}|}$$

In other words, the probability a random selection of variables is a root is proportional to the total degree and inversely proportional to the field size.

Proof. First consider the case $v = 1$, that is the univariate case. From field theory it is evident that a univariate polynomial of degree d has at most d roots. Thus

$$\Pr_{x \in \mathbb{F}}\big[f(x) = 0\big] \leq \frac{t}{|\mathbb{F}|}$$

Now for the sake of induction suppose the theorem holds for $v - 1$. Given a v-variate polynomial f, rearrange it around the monomials of the v'th variable such as

$$f(x_1, \ldots, x_v) = \sum_{i=0}^{t} g_i(x_1, \ldots, x_{v-1}) x_v^i \tag{3.1}$$

Since the degree of each term is at most t, we have

$$\deg(x_v^i) + \deg(g_i) \leq t \implies \deg(g_i) \leq t - i$$

Therefore by induction we have

$$\Pr_{x \in \mathbb{F}^{v-1}}\big[g_i(x_1, \ldots, x_{v-1}) = 0\big] \leq \frac{t - i}{|\mathbb{F}|}$$

Consider the component $g_t(x_1, \ldots, x_{v-1})$. We may condition whether or not $f(x_1, \ldots, x_v) = 0$ on whether or not $g_i(x_1, \ldots, x_{v-1}) = 0$. We already established that $g_i(x_1, \ldots, x_{v-1}) = 0$ with probability no more than $(t-i)/|\mathbb{F}|$. We now focus on the probability that $f(x_1, \ldots, x_v) = 0$ when $g_i(x_1, \ldots, x_{v-1}) \neq 0$. Observing the decompoisition in equation 3.1, we see this occurs with probability at most $i/|\mathbb{F}|$ by the original univariate case of the Schwartz-Zippel lemma which follows by induction. Therefore the probability $f(x_1, \ldots, x_v) = 0$ is bounded above as follows.

$$\Pr_{x \in \mathbb{F}^v} f(x_1, \ldots, x_v) = 0 \leq \frac{t - i}{|\mathbb{F}|} + \frac{i}{|\mathbb{F}|} = \frac{t}{|\mathbb{F}|}$$

This is what the lemma promised.

□

3.4.2 The evaluation reduction protocol

Using the Schwartz-Zippel lemma established above we now describe and prove the properties of the core protocol of both of our constructions. This is the building block of the sumcheck protocol. We refer to it as the **EvaluationReduction** protocol. This protocol seems to have been implicit in the original version of the sumcheck protocol presented in the pioneering work of [LFKN92]. The first explicit presentation of this protocol, however, seems to have appeared in [GKR15] It was not proven sound in [GKR15], however, and indeed we prove its soundness here. First we describe its construction and then its properties.

Construction

The purpose of the protocol is to reduce the task of verifying a function of multiple evaluations of a low degree multivariate polynomial to evaluating the polynomial at a single random point. We denote the polynomial by f, the variacy by v, the total degree by t, and the function of the evaluations by g. We assume there are $k \in \mathbb{N}$ evaluations. Specifically, suppose a prover would like to prove some function $g \in \mathbb{F}^k \to \mathbb{F}$ of $k \in \mathbb{N}$ evaluation claims at points $p_1, \ldots, p_k \in \mathbb{F}^v$ on a $(v \in \mathbb{N})$-variate polynomial $f \in \mathbb{F}[x_1, \ldots, x_v]$ of total degree $t \in \mathbb{N}$ over a field $\mathbb{F}$.

The prover and verifier engage as follows.

1. The prover and verifier both interpolate the univariate polynomials $h_1, \ldots, h_k \in \mathbb{F}[x]$ that pass through all k values $p_1, \ldots, p_k \in \mathbb{F}^v$. For simplicity we may choose the interpolation points to be the first k natural numbers $[k]$, assuming the field has characteristic at least k. If the field has characteristic less than k one may choose arbitrary elements in the extension field as one pleases.

   Specifically, the polynomials $h_1, \ldots, h_k \in \mathbb{F}[x]$ are uniquely interpolated with degree at most $k-1$ each such that

   $$\forall i \in [k],\ \forall j \in [v],\ h_j(i) = (p_i)_j$$

2. The prover performs the following evaluation defining $f'(x) \in \mathbb{F}[x]$ and sends the result to the verifier.

   $$f'(x) \coloneqq f(h_1(x), \ldots, h_v(x))$$

   Note that f' should be of degree at most $t(k-1)$. This is because each h_j for $j \in [v]$ is of degree at most $k-1$ and f has total degree at most t.

3. The verifier receives $f' \in \mathbb{F}[x]$ from the prover and checks that it has degree at most $t(k-1)$. Then the verifier evaluates it at the k interpolation points. In the case of the natural numbers, the verifier computs evaluations $e_1, \ldots, e_k \in \mathbb{F}$ as follows.

   $$\forall i \in [k],\ e_i = f'(i)$$

4. Using these evaluations previous computed the verifier now plugs them into the function g and checks whether they yield the claimed output value.

   $$g(e_1, \ldots, e_v)$$

5. Lastly the verifier must check that the polynomial f' sent by the prover is indeed correct. To do so the verifier takes advantage of the Schwarts-Zippel lemma. Specifically, the verifier will select a random point $\rho \in \mathbb{F}$ and perform identity testing on the following two polynomials, that is checking the following equality.

$$f'(\rho) = f(h_1(\rho), \ldots, h_v(\rho))$$

The verifier may immediately compute the former given that the prover sent it the function f'. In order to compute the latter, the verifier *does not* compute the composition of the h polynomials in f and then plug in ρ, as doing so would be doing the same as the prover, thus saving no cost. Instead, the verifier computes $h_1(\rho), \ldots, h_v(\rho)$ given its access to the interpolated h polynomials computed in the first step. Suppose $r_i = h_i(\rho)$ for $i \in [v]$. Then the verifier is left to compute $f(r_1, \ldots, r_v)$ and verify the following equality.

$$f'(\rho) = f(r_1, \ldots, r_v)$$

If this is a subprotocol and the verifier is to delegate the evaluation of $f(r_1, \ldots, r_v)$ to the prover, the verifier sends ρ as a return message.

Properties

The following theorem states and proves the properties of the previous constructed protocol.

Theorem 3 (EvaluationReduction protocol properties)**.** *The* ***EvaluationReduction*** *protocol described above is a public coin, interactive proof with the following properties.*

Completeness: *The completeness error is zero. That is, if the prover is honest, the prover succeed in proving with probability* 1*.*

Soundness: *The soundness error is no more than* $t(k-1)/|\mathbb{F}|$*. That is, if the prover is dishonest, the prover will succeed in proving with probability at most* $t(k-1)/|\mathbb{F}|$*.*

Round complexity: *Clearly, the above protocol consists of a single message from the prover to the verifier, perhaps followed by a message from the verifier to the prover if the protocol is a subprotocol inside a larger protocol. Therefore the protocol has round complexity at most* 1*.*

Proof. **Completeness**

One may observe completeness holds by observing associative relation between the polynomial f, the h polynomials, and the random value ρ. These three objects are composed together via function composition as follows.

$$f(h_1(\rho), \ldots, h_v(\rho))$$

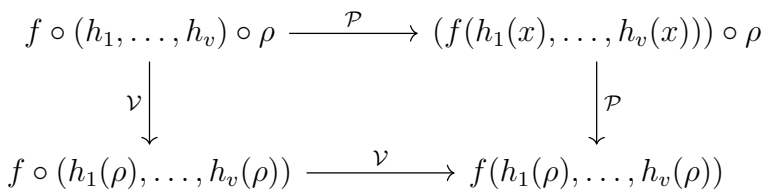

Figure 3.1: The associative paths travelled by $\mathcal{P}$ and $\mathcal{V}$ doing the evaluation reduction.

Function composition is an associative relation, thus the following are equivalent.

$$(f \circ (h_1, \ldots, h_v)) \circ \rho = (f(h_1(x), \ldots, h_v(x))) \circ \rho = f(h_1(\rho), \ldots, h_v(\rho))$$
$$= f \circ (h_1(\rho), \ldots, h_v(\rho)) = f \circ ((h_1, \ldots, h_v)) \circ \rho)$$

The prover computes them in the expensive direction, from left to right, first evaluating f on the h polynomials. The verifier computes the cheap direction, from right to left, first evaluating the h polynomials on ρ. The fact that the relation is associative is what allows the prover and verifier to arrive at the same conclusions, therefore completeness holds. We draw a commutative diagram in Figure-3 to illustrate this associative relation, in which we denote the prover by $\mathcal{P}$ and the verifier by $\mathcal{V}$.

Soundness:

Soundness follows from the Schwartz-Zippel lemma, in that the verifier's only randomized reduction employed in the protocol is to evaluate two polynomials at a random point to perform identity testing. As mentioned in the construction, both polynomials are of maximal degree $t(k-1)$, and therefore so is their difference. Suppose the polynomials are different. Then their difference is non-zero. The soundness error is the probability the two different polynomials evaluate the same on random input ρ, that is the probability their difference evaluates to 0 on the random input ρ. By the Schwartz-Zippel lemma, this occurs with probability at most $t(k-1)/|\mathbb{F}|$.

Round complexity: One may verify that round complexity is 1 by examining the construction.

□

3.4.3 The sumcheck protocol

Using the **EvaluationReduction** protocol recursively we construct the sumcheck protocol intended to verify the sum of many evaluations over a low degree polynomial. The sumcheck protocol was implicit in the construction of [LFKN92]. It has also been the core protocol use in numerous works, both theoretical and pratical. On the theoretical side it was used to prove both that IP = PSPACE and that MIP = NEXP in respective papers [Sha92] and [BFL91]. On the practical side, it was first used to delegate computations of the complexity class NC in [GKR15], after which many subsequent works followed.

Construction

Given a low degree v-variate polynomial $f\colon \mathbb{F}^v \to \mathbb{F}$ of total degree at most t suppose a prover would like to prove the summation of evaluations over a large evaluation set. In practice, the large evaluation set used is the boolean hypercube of dimension v, but any other set would work. For simplicity and relevance to how the sumcheck is used in this project, we focus on the boolean hypercube $\{0,1\}^v$. Following is the protocol such that a prover may prove to a verifier the value of

$$\sum_{x \in \{0,1\}^v} f(x_1, \ldots, x_v)$$

The prover and verifier execute the **EvaluationReduction** protocol on f for the two evaluation points $\{0,1\}$ and the function of the evaluations being addition. At the end of the **EvaluationReduction** protocol the verifier is left to compute the following for some random point $r_1 \in \mathbb{F}$.

$$\sum_{x \in \{0,1\}^{v-1}} f(r_1, x_1 \ldots, x_{v-1})$$

To do so the prover and verifier again invoke the **EvaluationReduction** protocol, this time with the polynomial $f(r_1, x_1 \ldots, x_{v-1})$ and the evaluation points $\{0,1\}$. Similarly, this leaves the verifier to compute the following for some random point $r_2 \in \mathbb{F}$

$$\sum_{x \in \{0,1\}^{v-2}} f(r_1, r_2, x_1 \ldots, x_{v-2})$$

This process continues for all $i \in [v]$ until at the end the verifier is left to verifier for random points $r_1, \ldots, r_v \in \mathbb{F}$

$$f(r_1, \ldots, r_v)$$

Properties

Theorem 4 (Sumcheck protocol properties)**.** *The sumcheck protocol described above is a public coin, interactive proof with the following properties.*

Completeness: *The completeness error is zero. That is, if the prover is honest, the prover succeeds in proving with probability* 1*.*

Soundness: *The soundness error is no more than $vt/|\mathbb{F}|$. That is, if the prover is dishonest, the prover will succeed in proving with probability at most $vt/|\mathbb{F}|$.*

Round complexity: *The round complexity is v.*

Proof. **Completeness:**

Completeness follows from the completeness of the **EvaluationReduction** protocol. In particular, the prover is able to maintain comleteness at each step of the sumcheck protocl via the promised completeness of the **EvaluationReduction** protocol.

Soundness:

Soundness follows from the soundness of the **EvaluationReduction** protocol. In particular, each round of the **EvaluationReduction** protocol incurs soundness error at most $t/|\mathbb{F}|$. This is because in each such invocation of the **EvaluationReduction** protocol the points of evaluation are the two points $\{0, 1\}$ and thus $k = 2$. By the union bound, we may sum the soundness errors from each round to obtain a total soundness error of $vt/|\mathbb{F}|$

Round complexity:

For each of the v variables the **EvaluationReduction** protocol must be invoked once. Since the **EvaluationReduction** protocol has round complexity 1, the sumcheck protocol has round complexity as most v.

□

Lastly we introduce some notation we will use in both the discrete log and lattice constructions. This notation concerns multivariate polynomials. Frequently in our construction and analysis we will be invoking multivariate polynomials on not just single-symbol variables, but variables composed of a combination of symbols. Therefore, often some of the variables come from a single symbol while others come individually from their own symbols. This is a recipe for confusion, not to mention the existing possiblity for confusion between a multivariate and univariate polynomial when invoked on a single symbol. Therefore we adopt the following notation, called 'spreading' notation which has recently become popular in programming languages. Suppose $f\colon \mathbb{F}^v \to \mathbb{F}$ is a v-variate polynomial and we wish to invoke it on input $x \in \mathbb{F}^v$. Instead of writing $f(x)$ we write $f(\ldots x)$. The idea is that the three condensed dots '…' function as a univariate operator that accepts a vector and 'spreads' its elements out in order such that they are in appropriate format for feeding to a multivariate function.

Chapter 4

Saga Batching in the Discrete Log Setting

4.1 Definitions

First we define the model of arithmetic programs we will use in our discrete log based construction. We will then define the notion of statement-witness pairs and what it means for one to satisfy such an arithemtic program.

Definition 12 (Arithmetic program in the discrete log setting)**.** *We define an* ***arithmetic program*** *in the discrete log setting* $(\mathbb{F}, H, \texttt{params}, A, B, C)$ *as consisting of the five parts in the previous tuple, categorized in three ways. In particular, the first part is the field together with a multiplicative subgroup* H*, the second part is a set of parameters, the third part is three field valued functions.*

- *The first part* $\mathbb{F}$ *is a finite field of prime order. The second part is a subset* $H \subseteq \mathbb{F}$*. Beyond just a subset of* $\mathbb{F}$*,* H *is a multiplicative subgroup of* $\mathbb{F}$*.*
- *The third part* `params` *is an ordered list of parameters, all natural numbers indicating a size.*
  1. `constraintCount` $\in \mathbb{N}$ *which we abbreviate* `Ccount`*.*
  2. `statementSize` $\in \mathbb{N}$ *which we abbreviate* `Ssize`*.*
  3. `traceSize` $\in \mathbb{N}$ *which we abbreviate* `Tsize`*.*
  4. `maskSize` $\in \mathbb{N}$ *which we abbreviate* `Msize`*.*
  5. `quotientSize` $\in \mathbb{N}$ *which we abbreviate* `Qsize`*.*
  6. `variableSize` $\in \mathbb{N}$ *which we abbreviate* `Vsize`*.*
  7. `witnessSize` $\in \mathbb{N}$ *which we abbreviate* `Wsize`*.*
  8. `solutionSize` $\in \mathbb{N}$*.*

  Often we implicitly access these internal parameters from `params` *rather than using explicit dot-access notation such as* `params.constraintCount`*.*

The last four parameters are formulated entirely dependent on the former parameters as follows.

$$\begin{aligned}
\texttt{quotientSize} &\coloneqq \texttt{constaintCount} - 1 \\
\texttt{variableSize} &\coloneqq 1 + \texttt{statementSize} + \texttt{traceSize} + \texttt{maskSize} \\
\texttt{witnessSize} &\coloneqq \texttt{traceSize} + 3 * \texttt{maskSize} + \texttt{quotientSize} \\
\texttt{solutionSize} &\coloneqq \texttt{statementSize} + \texttt{witnessSize}
\end{aligned}$$

We now mention an alias we will fequently use: we define $\ell \coloneqq \lceil \log(\texttt{ConstraintCount}) \rceil$.

Furthermore, we have the following parameter relationships with H*: the order of* H *is* $|H| = \texttt{constraintCount}$*, and therefore one may observe that* `constraintCount` *divides* $|\mathbb{F}| - 1$ *by the fact that* H *is a subgroup of the multiplicative group of* $\mathbb{F}$ *which has order* $|\mathbb{F}| - 1$.

- *The last three parts are functions referred to as* A, B, C *expressing the logic of the arithmetic program. The logic functions have the following signature:* $A, B, C \colon H \times [\texttt{variableSize}] \to \mathbb{F}$.

  For the sake of zero-knowledge we reserve a subset $H' \subseteq H$ *which will not correspond to actual constraints, but rather* 0 *and* 1 *values. In particular,* $|H'| = 3\texttt{Msize}$ *and we partition it into three parts* H'_A, H'_B, H'_C *such that*

$$\begin{aligned}
\forall h \in H'_A,\ \forall i \in \{\texttt{Vsize} - \texttt{Msize} + 1, \ldots, \texttt{Vsize}\},\ A(h,i) = 1 \\
\forall h \in H'_A,\ \forall i \in \{0, \ldots, \texttt{Vsize} - \texttt{Msize}\},\ A(h,i) = 0, \\
\forall h \in H'_B \cup H'_C, \forall i \in \{0, \ldots, \texttt{Vsize}\},\ A(h,i) = 0
\end{aligned}$$

  and likewise for H'_B *and* H'_C *relative to* B *and* C.

  The three functions A, B, C *can be thought of elements of* $\mathbb{F}^{|H| \times \texttt{variableSize}}$*, or as matrices over the field* $\mathbb{F}$ *with* $|H|$ *rows and* `variableSize` *columns. One may have an injective ordering function* $\iota \colon [|H|] \to H$ *inducing a correspondence between* H *and the row indices, such that element* (i, j) *of the matrix is equal to* $A(\iota(i), j)$.

Now we introduce what a statement and a witness are in terms of an arithmetic progam as defined above for the discrete log setting.

Definition 13 (Statement-witness pair)**.** *Given an arithmetic program* $(\mathbb{F}, H, \texttt{params}, A, B, C)$ *we define what a* ***statement-witness pair*** (s, w) *is with respect to the arithmetic program.*

Here we specify the structure of the statement-witness pair (s, w)*. The first part is the* ***statement*** *with type* $s \in \mathbb{F}^{\texttt{Ssize}}$*. The second part is the* ***witness*** $\mathrm{w} = (t, \mathrm{m}, q)$ *with type a tuple constructed of three parts which we now describe.*

- *The* ***trace*** t *is a vector of field elements* $t \in \mathbb{F}^{\texttt{Tsize}}$.
- *The* ***mask*** m *is a triple* $\mathrm{m} = (m_A, m_B, m_C)$ *with each part a vector of field elements* $m_A, m_B, m_C \in \mathbb{F}^{\texttt{Msize}}$.

- *The **quotient** q is vector of field elements $q \in \mathbb{F}^{\texttt{Qsize}}$ representing a polynomial. Specifically, they represent the coefficients of a polynomial $\sum_{i=1}^{\texttt{Qsize}} q_i x^{i-1}$ of degree less than* `Qsize` *which we refer to as the **quotient polynomial**. Often we sligntly abuse notation and refer to q as both a vector and a polynomial, but with differing notations. When interpreted as a vector we access the i'th element as q_i, whereas when interpreted as polynomial on input x we write $q(x)$.*

Now we describe three important auxiliary notions with corresponding notations we will use frequently, that is the notions of variable vectors, the witness vector, and the solution vector. It is precisely for these notions that we previously defined the parameters `variableSize`*,* `witnessSize`*, and* `solutionSize`*.*

- *Here we define the notion of **variable vectors**, specifically the three vectors of* `variableSize` *variables $v_A, v_B, v_C \in \mathbb{F}^{\texttt{Vsize}}$. They may each be derived from the witness, and often we will refer to them directly. We define them via their canonical product-isomorphisms as*

$$v_A \cong (1, s, t, m_A) \text{ and } v_B \cong (1, s, t, m_B) \text{ and } v_C \cong (1, s, t, m_C)$$

where each left side is of type $\mathbb{F}^{\texttt{Vsize}}$ and each right side is of type $\mathbb{F} \times \mathbb{F}^{\texttt{Ssize}} \times \mathbb{F}^{\texttt{Tsize}} \times \mathbb{F}^{\texttt{Msize}}$.

- *Here we define the notion of the **witness vector** which is distinct from that of the witness $\mathbb{w}$. Like the variable vectors, we define it via a canonical isomorphism as follows.*

$$w \in \mathbb{F}^{\texttt{Wsize}} \cong (t, (m_A, m_B, m_C), q) \in \mathbb{F}^{\texttt{Tsize}+3*\texttt{Msize}+\texttt{Qsize}}$$

- *Here we define the notion of the **solution vector** or the **solution** which consists of the statement together with the witness vector, again by the following canonical isomorphism.*

$$z \in \mathbb{F}^{\texttt{Ssize}+\texttt{Wsize}} \cong (s, t, (m_A, m_B, m_C), q) \in \mathbb{F}^{\texttt{Ssize}+\texttt{Tsize}+3*\texttt{Msize}+\texttt{Qsize}}$$

The combination of the definition 12 of an arithmetic program in the discrete log setting along with the definition 13 of what a statement-witness pair is with respect to one, we have still not defined exactly what it means for such a statement-witness pair to satisfy the said arithemtic program. We do so now.

Definition 14 (Arithmetic program satisfaction)**.** *We now define what it means for a **statement-witness pair** $(s, \mathbb{w})$ **to satisfy an arithmetic program** $(\mathbb{F}, H, \texttt{params}, A, B, C)$.*

*To do so we introduce three function f_A, f_B, f_C which we refer to as the **polynomial-inducing functions** for A, B, and C respectively. They have the signature $f_A, f_B, f_C \colon \mathbb{F}^{\texttt{Vsize}} \times \mathbb{F} \to \mathbb{F}$ and are appropriately referred to as polynomial-inducing because upon plugging in a variable vector $v \in \mathbb{F}^{\texttt{Vsize}}$ as the first argument the result is a polynomial over $\mathbb{F}$. When polynomial-inducing functions are invoked on variable vectors, we refer to the resulting polynomials as **variable-induced polynomials**.*

We now define the polynomial-inducing functions. Specifically, for variables $v \in \mathbb{F}^{\texttt{Vsize}}$ we define the polynomials $f_A(v, x), f_B(v, x), f_C(v, x) \in \mathbb{F}[x]$ as the unique polynomials of degree

less than `constraintCount` *that satisfy the following interpolations over the set* H *of size* `constraintCount`*. For all* h *in* H *we demand that*

$$f_A(v,h) = \sum_{j=1}^{\texttt{Vsize}} A(h,j)v_j$$
$$f_B(v,h) = \sum_{j=1}^{\texttt{Vsize}} B(h,j)v_j$$
$$f_C(v,h) = \sum_{j=1}^{\texttt{Vsize}} C(h,j)v_j$$

With the functions f_A, f_B, f_C *in hand, we say the statement-witness pair* $(s, \mathbb{w})$ *satisfies the arithmetic program* $(\mathbb{F}, H, \texttt{params}, A, B, C)$ *if and only if the following condition is satisfied. Let* $\mathbb{w} = (t, (m_A, m_B, m_C), q)$ *with corresponding variable vectors* $v_A, v_B, v_C \in \mathbb{F}^{Vsize}$*. We require that the following assertions holds.*

$$(\forall h \in H)\ f_A(v_A,h)f_B(v_B,h) + f_C(v_C,h) = 0$$

The purpose of q *is to replace the above set of field equations with a single polynomial equation*. The above assertion holds if and only if* $q \in \mathbb{F}^{\texttt{Qsize}}$ *can be chosen such that the following alternative assertion holds.*

$$f_A(v_A,x)f_B(v_B,x) + f_C(v_C,x) = \left(\sum_{i=1}^{\texttt{Qsize}} q_i x^{i-1}\right)\left(\prod_{h\in H}(x-h)\right) \tag{4.1}$$

We clarify immediately below why these two assertions are equivalent. In brief, we say $(s, \mathbb{w})$ *satisfies the arithmetic program if and only if the latter polynomial equation holds with respect to the quotient polynomial* q *contained in the witness* $\mathbb{w}$*.*

We immediately move to proving the equivalence between to two statements of arithmetic program presented at the end of the previous definition 14.

Proposition 4 (Equivalence of satisfaction assertions)**.** *Borrowing the notational context from the preceding definition, we that prove the following set of field equations*

$$(\forall h \in H)\ f_A(v_A,h)f_B(v_B,h) + f_C(v_C,h) = 0$$

holds if and only if the following polynomial equation holds.*

$$(\exists q \in \mathbb{F}^{\texttt{Qsize}})\ f_A(v_A,x)f_B(v_B,x) + f_C(v_C,x) = \left(\sum_{i=1}^{\texttt{Qsize}} q_i x^{i-1}\right)\left(\prod_{h\in H}(x-h)\right)$$

Furthermore, one may replace $\prod_{h\in H}(x-h)$ *with* $x^{\texttt{Ccount}} - 1$*.*

Proof. Suppose the polynomial equation* holds. Then upon evaluating the left side of the field equation* at every value $h \in H$ the right side vanishes because h is a root of the polynomial $\prod_{h\in H}(x-h)$, called the 'polynomial vanishing on H.'

Conversely, suppose the set of field equations holds. Then H is a subset of the roots of the polynomial $f(x) \coloneqq f_A(v_A, x)f_B(v_B, x) + f_C(v_C, x)$. Therefore each $(x - h)$ for $h \in H$ is a factor of $f(x)$, so the polynomial $\prod_{h\in H}(x - h)$ is a factor of $f(x)$. We may then divided $f(x)$ by $\prod_{h\in H}(x-h)$ to obtain another polynomial. Left is to verify that $f(x)/\prod_{h\in H}(x-h)$ is indeed a polynomial of degree less than $\texttt{quotientSize}$.

Recall that $f_A(v_A, x)$, $f_B(v_B, x)$, and $f_C(v_C, x)$ are univariate polynomials of degree at most $\texttt{constraintCount} - 1$. Then the polynomial $f_A(v_A, x)f_B(v_B, x)+f_C(v_C, x)$ is of degree at most $2(\texttt{constraintCount} - 1)$. The polynomial $\prod_{h\in H}(x - h)$ vanishing on H has degree exactly $|H| = \texttt{constraintCount}$. Therefore $f(x)/\prod_{h\in H}(x - h)$ has degree at most

$$\begin{aligned} &2(\texttt{constraintCount} - 1) - \texttt{constraintCount} \\ &= \texttt{constraintCount} - 2 = \texttt{quotientSize} - 1 \end{aligned}$$

Thus one may set q to the $\texttt{quotientSize}$ coefficients of this quotient polynomial.

Regarding the equivalence between $\prod_{h\in H}(x-h)$ with $x^{\texttt{Ccount}}-1$, first note that $\texttt{Ccount} = |H|$. Since H is a multiplicative subgroup, all elements in H raised to the order of H equal the identity element 1. Therefore, any root of the polynomial $\prod_{h\in H}(x - h)$ is also a root of the polynomial $x^{\texttt{Ccount}} - 1$. Combining this with the fact that both polynomials are of equal degree $\texttt{Ccount} = |H|$, the two polynomials must have the same set of roots. Additionally noting they are both monic polynomials, they must equivalent. □

Lastly, we introduce an importnat correspondence we will use frequently throughout our construction. The correspondence relates univariate polynomials to multilinear polynomials. A multilinear polynomial is one multiple variables and linear in each variable. We use mutlilinear polynomials due to their minimal degree relative to the number of non-zero coefficients they may have. The purpose of the correspondence is to convert the induced univariate polynomials and the quotient polynomial into multivariate polynomials afte they are check with respect to the quadratic satisfaction equation. This way we may subsequently invoke an evaluation reduction protocol on them due to their low degree, similar to the protocol of 3.4.2.

Definition 15 (Multilinear polynomials)**.** *Previously we introduced the variable-induced univariate polynomials $f_A(v, x)$, $f_B(v, x)$, and $f_C(v, x)$ for some variable vector $v \in \mathbb{F}^{\texttt{Vsize}}$, all of degree less than* $\texttt{constraintCount}$*. We also introduced the concept of a quotient polynomial $q \in \mathbb{F}[x]$ as a univariate polynomial of degree less than* $\texttt{Qsize}$*. Now we introduce their multilinear versions $\widehat{f_A}(v, x)$, $\widehat{f_B}(v, x)$, $\widehat{f_C}(v, x)$, and $\widehat{q}(x)$.*

We intend to construct the multilinear versions such that upon plugging in a particular input we recover the same output as the univariate version. To do so, the simply replace the monomials of the univariate with the monomials of the multilinear by a natural bijection. Suppose we use univariate variable $x \in \mathbb{F}$ for a univariate polynomial of degree less than $2^d \in \mathbb{N}$. Then we define the correspondence between a univariate polynomial $f \in \mathbb{F}[x]$ and its d-variate multilinear version $f \in \mathbb{F}[x_1, \ldots, x_d]$ as follows.

$$f(x) = f(1, x, x^2, \ldots, x^d)$$

In other words, monomial x^i for $i \in \mathbb{N}$ of the univariate version is replaced with monomial $x_1^{b_1} \cdots, x_d^{b_d}$ where b_j is the j'th bit of the binary representation of i.

4.2 Construction

Let $(\mathcal{P}, \mathcal{V})$ be a pair of algorithms we refer to as the prover and verifier. The saga batching solution is with respect to an arithmetic program in the describe log setting $\mathbb{ap} = (\mathbb{Z}_p, H, \mathtt{params}, A, B, C)$ as defined in 12. We will refer to the internal parameters of $\mathbb{ap}$ directly, along with all the notions defined in 13, 14, and 15.

The prover and verifier will engage in a setup phase, followed by an initial phase. Then they will engage in $n = \text{poly}(\lambda)$ intermediate phases each phase corresponding to an independent proof. Finally they will engage in a final phase upon which the verifier will make a decision of acceptance or rejection. We will describe the setup of the protocol along with the protocol phases $i = 0 \ldots, n+1$. Phase 0 is called the initial phase, phases $1, \ldots, n$ are called the intermediate phases, and phase $n+1$ is called the final phase. The prover and verifier engage in the following sequence of protocols, the verifier potentially rejecting during at any time if expectations are not met.

1. $(\mathcal{P}, \mathcal{V})$ execute **Setup**$(\mathbb{ap})$
2. $(\mathcal{P}, \mathcal{V})$ execute **InitialPhase**$(\mathbb{ap}, g \in \mathbb{G}^{\mathtt{Wsize}})$
3. $(\mathcal{P}, \mathcal{V})$ execute **IntermediatePhase**$(i, \mathbb{ap}, \mathbb{cd}^{(i)}, \mathbb{pd}^{(i)})$ for $i \in [n]$ sequentially.
4. $(\mathcal{P}, \mathcal{V})$ execute **FinalPhase**$(\mathbb{ap}, \mathbb{cd}^{(n+1)}, \mathbb{pd}^{(n+1)})$
5. $\mathcal{V}$ accepts or rejects all n proofs in the batch.

For each intermediate phase $i \in [n]$ which concerns the statement-witness pair $(s, \mathbb{w})^{(i)}$, there will be many symbols involved in the construction below in addition to the symbols concerned with the arithmetic program or the statement-witness pair. We mention them now.

$$
\begin{aligned}
b_i, &\in_R \mathbb{Z}_p \\
\alpha_i, \beta_i, \rho_i &\in_R \mathbb{Z}_p \\
r_1^{(i)}, \ldots, r_\ell^{(i)} &\in \mathbb{Z}_p \\
e_A^{(i)}, e_B^{(i)}, e_C^{(i)}, e_q^{(i)} &\in \mathbb{Z}_p \\
\gamma_1^{(i)}, \ldots, \gamma_\ell^{(i)} &\in \mathbb{F}_p^{<2}[x] \\
\phi_A^{(i)}, \phi_B^{(i)}, \phi_C^{(i)}, \phi_q^{(i)} &\in \mathbb{Z}_p^{<\ell+1}[x] \\
\chi_A^{(i)}, \chi_B^{(i)}, \chi_C^{(i)}, \chi_q^{(i)} &\in \mathbb{Z}_p^{<\ell}[x] \\
\psi_A^{(i)}, \psi_B^{(i)}, \psi_C^{(i)}, \psi_q^{(i)} &\in \mathbb{Z}_p^{<\ell+1}[x]
\end{aligned}
$$

We write $\mathbb{Z}_p^{<d}[x]$ to denote the set of univariate polynomials over $\mathbb{Z}_p$ of degree less than d. We also clarify a parameter which will be use frequently enough it deserves an alias: $\ell \coloneqq \lceil \log(\mathtt{Ccount}) \rceil$.

4.2.1 Setup

The setup protocol **Setup**($\mathbb{ap}$) takes an arithmetic program for the discrete log setting $\mathbb{ap} = (\mathbb{Z}_p, H, \texttt{params}, A, B, C)$ and outputs the cryptographic constants defining the commitment scheme.

The prover and verifier $(\mathcal{P}, \mathcal{V})$ agree on a uniformly selected vector of $\texttt{witnessSize} + 1$ elements from a group $\mathbb{G}$ of order $|\mathbb{G}| = p$ in which the discrete log problem is hard. The extra constant beyond the witness size is for commitment masks in regard to zero-knowledge. We denote this vector by $g \in \mathbb{G}^{\texttt{Wsize}+1}$. The setup phase outputs $(\mathbb{ap}, g)$.

4.2.2 Initial phase

The initial phase protocol **InitialPhase**($\mathbb{ap}, g \in \mathbb{G}^{\texttt{Wsize}}$) accepts the arithmetic program $\mathbb{ap}$ and the vector of group elements defining the Petersen commitment scheme $g \in \mathbb{G}^{\texttt{Wsize}}$. The initial phase will end with the prover sending the verifier data that defines $\mathbb{cd}^{(1)}$.

The initial phase consists of a single message from the prover to the verifier. The prover executes the following sequence of steps to compute the message and then send it.

1. The prover selects vector $s'^{(0)} \in \mathbb{Z}_p^{\texttt{Ssize}}$ and uniform vector $w'^{(0)} \in_R \mathbb{Z}_p^{\texttt{Wsize}}$ which we may think of as a fake statement and a fake witness respectively. The fake statement can be anything, such as the zero vector $[0]_{i \in \texttt{Ssize}}$. The fake witness must be uniform because it is for the purpose of zero-knowledge in that it will shield against all of the coming n real witness vectors.

2. The prover computes the commitment $c'^{(0)}$ to the vector $w'^{(0)}$ by choosing a random commitment mask $b' \in_R \mathbb{Z}_p$ and computing

$$c'^{(0)} = g_{\texttt{Wsize}+1}^{b'} \prod_{j \in [\texttt{Wsize}]} g_j^{w_j'^{(0)}}$$

3. Given the fake witness vector $w'^{(0)}$, the prover constructs fake variable vectors $v_A'^{(0)}, v_B'^{(0)}, v_C'^{(0)} \in \mathbb{Z}_p^{Vsize}$ and from them constructs the induced polynomials $f_A(v_A'^{(0)}, x), f_B(v_B'^{(0)}, x), f_C(v_C'^{(0)}, x)$. The prover also uses the fake witness to construct the fake quotient polynomial $q'^{(0)} \in \mathbb{Z}_p[x]$. Now the prover evaluates them all at some point $r^{(0)}$ and records their evaluations $e_A^{(0)}, e_B^{(0)}, e_C^{(0)}, e_q^{(0)}$. The input to evaluation $r^{(0)}$ may be arbitrary, and for simplicity perhaps fixed to $0 \in \mathbb{Z}_p^\ell$ in which case the prover sets them to

$$\begin{aligned} e_A^{(0)} &\coloneqq f_A(v_A'^{(0)}, \ldots [0]_{i \in [\ell]}) \\ e_B^{(0)} &\coloneqq f_B(v_B'^{(0)}, \ldots [0]_{i \in [\ell]}) \\ e_C^{(0)} &\coloneqq f_C(v_B'^{(0)}, \ldots [0]_{i \in [\ell]}) \\ e_q^{(0)} &\coloneqq q'^{(0)}(\ldots [0]_{i \in [\ell]}) \end{aligned}$$

4. The prover sends the verifier the commitment $c'^{(0)}$ and the four evaluations $e_A^{(0)}, e_B^{(0)}, e_C^{(0)}, e_q^{(0)}$. The communication cost is 1 group element and 4 field elements.

Suppose we assume $s'^{(0)}$ and $r^{(0)}$ are both predetermined, such as their zero-variants mentioned above. Then upon completion of the above procedure both the prover and verifier have the data necessary to proceed to phase 1. Specifically, both parties have $\mathbb{cd}^{(1)}$ which as explained below denotes the common data held in by both the prover an verifier when phase 1 begins, defined as follows.

$$\mathbb{cd}^{(1)} := (s'^{(0)}, c'^{(0)}, r^{(0)}, (e_A^{(0)}, e_B^{(0)}, e_C^{(0)}, e_q^{(0)}))$$

The prover also has the pseudo witness vector $w'^{(0)}$ necessary for phase 1.

4.2.3 Intermediate phases

The intermediate phase protocols $\textbf{IntermediatePhase}(i, \mathbb{ap}, \mathbb{cd}^{(i)}, \mathbb{pd}^{(i)})$ for $i \in [n]$ consist of the phase number $i \in \mathbb{N}$, the arithmetic program $\mathbb{ap}$, the tuple $\mathbb{cd}^{(i)}$ containing the common data held by both the prover and the verifier when phase i begins, and the tuple $\mathbb{pd}^{(i)}$ containing the additional data held by the prover (not known to the verifier) when phase i begins. We define these tuples as follows and then describe their contents.

$$\mathbb{cd}^{(i)} = (s'^{(i-1)}, c'^{(i-1)}, r^{(i-1)}, (e_A^{(i-1)}, e_B^{(i-1)}, e_C^{(i-1)}, e_q^{(i-1)}))$$
$$\mathbb{pd}^{(i)} = (w'^{(i-1)}, (s, \mathbb{w})^{(i)})$$

We refer to $s'^{(i-1)} \in \mathbb{Z}_p^{Ssize}$ as a 'pseudo statement' because it is the random combination of actual statements sent by the prover. We refer to $c'^{(i-1)} \in \mathbb{G}$ as a 'pseudo commitment' because although it is a true commitment, it is a random combination of commitments sent by the prover. The prover supposedly has a vector $w'^{(i-1)} \in \mathbb{Z}_p^{\texttt{Wsize}}$ binded to the commitment $c'^{(i-1)}$ which we refer to as the 'pseudo-witness' because again it is the random combination of real witnesses. Often we will refer to the pseudo variable vectors $v_A'^{(i-1)}, v_B'^{(i-1)}, v_C'^{(i-1)}$ corresponding to $w'^{(i-1)}$. The point $r^{(i-1)} \in \mathbb{Z}_p^{\ell}$ is an input to the polynomials induced by $v_A'^{(i-1)}, v_B'^{(i-1)}, v_C'^{(i-1)}$ and the quotient polynomial $\widehat{q'^{(i-1)}}$supposedly resulting in evaluations $e_A^{(i-1)}, e_B^{(i-1)}, e_C^{(i-1)}, e_q^{(i-1)} \in \mathbb{Z}_p \backslash H$. That is, the prover claims that the following, and the verifier's acceptance of proofs so far is conditioned on the following.

$$\widehat{f_A}(v_A'^{(i-1)}, ...r^{(i-1)}) = e_A^{(i-1)}$$
$$\widehat{f_B}(v_B'^{(i-1)}, ...r^{(i-1)}) = e_B^{(i-1)}$$
$$\widehat{f_A}(v_C'^{(i-1)}, ...r^{(i-1)}) = e_C^{(i-1)}$$
$$\widehat{q'^{(i-1)}}(...r^{(i-1)}) = e_q^{(i-1)}$$

Lastly, the prover has a new statement-witness pair $(s, \mathbb{w})^{(i)}$ and the goal of phase i is proving that it is a valid statement-witness pair. Actually, phase i will not in fact prove that $(s, \mathbb{w})^{(i)}$ is valid, but instead amortize proving $(s, \mathbb{w})^{(i)}$ together with proving that the previous equations hold.

Each intermediate phase consists of 5 steps. In the first step the prover sends a message to the verifier, followed by the second step in which the verifier sending back a challenge message. In the third step both the prover and the verifier separately compute data needed

for the fourth and fifth steps. In the fourth step the prover sends a message to the verifier, followed by the fifth step in which the verifier sending back a challenge message. The first and second step together constitute a round, and so do the fourth and fifth steps. Thus intermediate phases each consist of 2 rounds.

At the end of phase i the we have common data necessary to proceed to phase $i+1$. The data includes the pseudo witness $\mathbb{w}'^{(i)}$ known to the prover, and the common data $\mathbb{cd}^{(i+1)}$ known to both parties.

$$\mathbb{cd}^{(i+1)} = (s'^{(i)}, c'^{(i)}, r^{(i)}, (e_A^{(i)}, e_B^{(i)}, e_C^{(i)}, e_q^{(i)}))$$

Step 1:

$(\mathcal{P} \to \mathcal{V})$:

Since the prover already has a witness $\mathbb{w}^{(i)}$, it should include a quotient polynomial q, but if the prover has only solved the trace and selected masks the quotient polynomial $q^{(i)}$ may be computed as

$$q^{(i)} \coloneqq \big(f_A(v_A, x) f_B(v_B, x) + f_C(v_C, x)\big) \Big/ \big(x^{\texttt{Ccount}} - 1\big)$$

Then the prover commits to the witness vector $w_j^{(i)} \in \mathbb{Z}_p^{\texttt{Wsize}}$ by choosing a random commitment mask $b_i \in \mathbb{Z}_p$ and computing

$$c^{(i)} \coloneqq g_{\texttt{Wsize}+1}^{b_i} \prod_{j \in [\texttt{Wsize}]} g_j^{w_j^{(i)}}$$

The prover sends the statement $s^{(i)} \in \mathbb{Z}_p^{\texttt{Ssize}}$ and the witness commitment $c^{(i)} \in \mathbb{G}$ to the verifier. The communication cost is 1 group element and `Ssize` field elements.

Step 2:

$(\mathcal{V} \to \mathcal{P})$:

The verifier samples a random challenge $\alpha_i \in_R \mathbb{Z}_p$. The purpose of α_i is to check that the equation corresponding to 4.1 by means of the Schwartz-Zippel lemma using α_i as the random element on which to perform the identity testing. The verifier sends α_i to the prover, the communication cost being 1 field element. The randomness complexity is also 1 field element.

Step 3:

($\mathcal{P}$ and $\mathcal{V}$ compute independently) : This step involves not interaction between the prover and verifier. Instead they each compute the following line $\gamma^{(i)}$. The line $\gamma^{(i)}$ is actually an ℓ-dimensional line, that is $\gamma^{(i)} \in (\mathbb{Z}_p[x])^\ell$ where $\gamma_j^{(i)} \in \mathbb{Z}_p[x]$ is a univariate polynomial of degree at most one. We write

$$\gamma^{(i)} = [\gamma_j^{(i)} \in \mathbb{Z}_p[x]]_{j \in [\ell]}$$

The multidimensional line γ is intended to be a connecting line between the two ℓ-dimensional points $r^{(i-1)} \in \mathbb{Z}_p^\ell$ and $[1, \alpha_i, \alpha_i^2, \ldots, \alpha_i^{2^{\ell-1}}] \in \mathbb{Z}_p^\ell$. One input to $\gamma^{(i)}$ will output the first point,

and a second input will output the second point. We may choose any two distinct inputs for this purpose, but for simplicity we choose 0 for the first input and 1 for the second input. This means we will have

$$\gamma^{(i)}(0) = r^{(i-1)} \text{ and } \gamma^{(i)}(1) = [\alpha_i^{2^{j-1}}]_{j\in[\ell]}$$

One may explicitly construct each component $\gamma_j^{(i)}$ for $j \in [\ell]$ of $\gamma^{(i)}$ as follows.

$$\gamma_j^{(i)}(x) = (1-x)r_j^{(i-1)} + x \cdot \alpha_i^{2^{j-1}}$$

Step 4:

$(\mathcal{P} \to \mathcal{V})$:

At this point in the i'th phase the prover computes the four $\phi^{(i)}$ polynomials and the four $\chi^{(i-1)}$ polynomials. The purpose is two fold and we have merged them together for efficiency, lower communication cost, and simplifying notation to avoid introducing additional symbols. If we did not blend the two purposes together but kept them distinct we could execute the following two steps protocol below, which we call the *hypothetical* protocol because we do not execute it. Each step would roughly proceed as follows.

In the first step the prover would use the random challenge α_i to prove quotient polynomial $q^{(i)}$ is consistent with the the three induced polynomials $f_A(v_A^{(i)}), f_A(v_A^{(i)}), f_A(v_A^{(i)})$ such that the $(s, \mathbb{w})^{(i)}$ for a valid statement-witness pair. To order to prove equation 4.1, the prover evaluates the three induced polynomials and the quotient polynomial all at α_i. The prover sends the four evaluations to the verifier, who then checks they satisfy the necessary equation which also involves $\prod_{h\in H}(\alpha_i - x)$.

In the second step the purpose it to confirm that the four evaluations sent by the prover are correct. The verifier, however, is already tasked with verifying the following four evaluations.

$$\begin{aligned}
e_A^{(i-1)} &= \widehat{f_A}(v_A'^{(i-1)}, \ldots r'^{(i-1)}) \\
e_B^{(i-1)} &= \widehat{f_B}(v_B'^{(i-1)}, \ldots r'^{(i-1)}) \\
e_C^{(i-1)} &= \widehat{f_C}(v_C'^{(i-1)}, \ldots r'^{(i-1)}) \\
e_q^{(i-1)} &= \widehat{q}(\ldots r'^{(i-1)})
\end{aligned}$$

So actually purpose is to amortize the verification of these four evaluations with the verification of the new four evaluations. The four evaluations above are with respect to the multilinear versions of the induced polynomials defined in definition 15 with multivariate input $r^{(i-1)}$, whereas the four new evaluations are with respect to the univariate versions of the induced polynomials with univariate input α_i. In order to amortize them we must first switch our interpretation of the induced polynomials from the univariate version to the multivariate version. Then we use the line $\gamma^{(i)}$ and four executions of the **EvaluationReduction** protocol to reduce from verifying the two sets of induced and quotient multilinear polynomials at the two distinct multivariate inputs $r^{(i-1)}$ and $(1, \alpha_i, \alpha_i^2, \ldots, \alpha_i^{\ell-1})$ to verifying the two sets of multilinear polynomials at a single multivariate input. Then we take a random combination of the two sets of evaluations with respect to the single multivariate input, thus

reducing from verification of two sets of multilinear polynomials to verification of one set of multilinear polynomials. One may consult definition 15 or protocol **EvaluationReduction** (3.4.2) to further understand the ideas involved.

In our construction, however, we do not employ the two step hypothetical protocol. Instead we use the two step protocol described in this step and the next step of phase i. We blend the two steps together in for optimization reasons and notational convenience. For example, if we used the **EvaluationReduction** protocol we'd need to execute it four times, whereas below we execute a generalized version of it only once. Also, each execution of the **EvaluationReduction** protocol would be wasting because the prover would be sending more information than necessary. This is because the verifier would already know evaluations on the two multivariate inputs, and evaluation on the line connecting them encodes those two evaluations as well, thus a redundancy. We avoid this redundancy merging the two steps as this and the next step of the protocol illustrate.

The prover computes the following four $\phi^{(i)}$ univariate polynomials. Note that upon plugging in 1 for x the result is the four evaluations the prover would have sent in step one of the hypothetical protocol. Thus by sending these polynomials to the verifier the prover is implicitly starting execution of step one.

$$\begin{aligned}
\phi_A^{(i)}(x) &\coloneqq \widehat{f_A}\big(v_A^{(i)}, ...\gamma^{(i)}(x)\big) \\
\phi_B^{(i)}(x) &\coloneqq \widehat{f_B}\big(v_B^{(i)}, ...\gamma^{(i)}(x)\big) \\
\phi_C^{(i)}(x) &\coloneqq \widehat{f_C}\big(v_C^{(i)}, ...\gamma^{(i)}(x)\big) \\
\phi_q^{(i)}(x) &\coloneqq \widehat{q^{(i)}}\big(...\gamma^{(i)}(x)\big)
\end{aligned} \tag{4.2}$$

The first three are of degree less than $\ell+1$ and the last is of degree less than ℓ. Thus sending these polynomials to the verifier will cost $3(\ell+1)+\ell = 4\ell+3$ field elements.

The prover also computes the following four $\chi^{(i-1)}$ univariate polynomials. Note that if we have not subtracted the four evaluations $e_A^{(i-1)}, e_A^{(i-1)}, e_A^{(i-1)}, e_A^{(i-1)}$ and divided by x, then upon plugging in 0 for x the result would be precisely those four evaluations, which the verifier already knows. The subtraction and division take advantage of the verifier's knowledge and only send what is necessary for communication efficiency.

$$\begin{aligned}
\chi_A^{(i-1)}(x) &\coloneqq \Big(\widehat{f_A}\big(v_A'^{(i-1)}, ...\gamma^{(i)}(x)\big) - e_A^{(i-1)}\Big)\Big/x \\
\chi_B^{(i-1)}(x) &\coloneqq \Big(\widehat{f_B}\big(v_B'^{(i-1)}, ...\gamma^{(i)}(x)\big) - e_B^{(i-1)}\Big)\Big/x \\
\chi_C^{(i-1)}(x) &\coloneqq \Big(\widehat{f_C}\big(v_C'^{(i-1)}, ...\gamma^{(i)}(x)\big) - e_C^{(i-1)}\Big)\Big/x \\
\chi_q^{(i-1)}(x) &\coloneqq \Big(\widehat{q'^{(i-1)}}\big(...\gamma^{(i)}(x)\big) - e_q^{(i-1)}\Big)\Big/x
\end{aligned}$$

The first three polynomials are of degree less than ℓ and the last polynomial is of degree less than $\ell-1$. Thus sending these polynomials to the verifier will cost $3\ell+(\ell-1) = 4\ell-1$ field elements. In total the prover sends eight univariate polynomials to the verifier, represented by $8\ell+2$ field elements.

Step 5:

$(\mathcal{V} \to \mathcal{P})$:

Now the verifier uniformly chooses an input $\beta_i \in_R \mathbb{Z}_p$ to the line $\gamma^{(i)}$. The verifier now computes the new multivariate input $r^{(i)}$ as

$$r^{(i)} = \left[r_j^{(i)}\right]_{j\in[\ell]} \coloneqq \left[\gamma_j^{(i)}\left(\beta^{(i)}\right)\right]_{j\in[\ell]} \in \mathbb{Z}_p^{\ell} \tag{4.3}$$

By sending the $\chi^{(i-1)}$ and $\phi^{(i)}$ polynomials the prover has implicitly made claims about the evaluations of the two sets of induced polynomials and the quotient polynomials at input $r^{(i)}$.

The verifier recovers these implicit evaluations and checks they satisfy the arithmetic program satisfaction equation 4.1.

$$\phi_A^{(i)}(1)\cdot\phi_B^{(i)}(1) + \phi_C^{(i)}(1) = \phi_q^{(i)}(1)\left(\alpha_i^{\texttt{Ccount}} - 1\right)$$

Next the verifier uses the four $\chi^{(i-1)}$ polynomials previously sent by the prover and reconstructs them into four $\psi^{(i-1)}$ polynomials the prover would have sent in the second step of the hypothetical protocol.

$$\begin{aligned}
\psi_A^{(i-1)}(x) &\coloneqq x\cdot\chi_A^{(i-1)}(x) + e_A^{(i-1)} \\
\psi_B^{(i-1)}(x) &\coloneqq x\cdot\chi_B^{(i-1)}(x) + e_B^{(i-1)} \\
\psi_C^{(i-1)}(x) &\coloneqq x\cdot\chi_C^{(i-1)}(x) + e_C^{(i-1)} \\
\psi_q^{(i-1)}(x) &\coloneqq x\cdot\chi_q^{(i-1)}(x) + e_q^{(i-1)}
\end{aligned} \tag{4.4}$$

The verifier uniformly chooses an field element $\rho_i \in_R \mathbb{Z}_p$ which it will employ in the random combination of the two sets of multilinear polynomials. To finally complete step two of the hypothetical protocol and reduce the two sets of evaluations to one set of evaluations, the verifier plugs in β_i to the $\psi^{(i-1)}$ and $\phi^{(i)}$ polynomials to obtain the implicitly claimed evaluations at $r^{(i)}$ and then takes a random combination of them to obtain final claimed evaluations $e_A^{(i)}, e_B^{(i)}, e_C^{(i)}, e_q^{(i)}$.

$$\begin{aligned}
e_A^{(i)} &\coloneqq \psi_A^{(i-1)}(\beta_i) + \rho_i\cdot\phi_A^{(i)}(\beta_i) \\
e_B^{(i)} &\coloneqq \psi_B^{(i-1)}(\beta_i) + \rho_i\cdot\phi_B^{(i)}(\beta_i) \\
e_C^{(i)} &\coloneqq \psi_C^{(i-1)}(\beta_i) + \rho_i\cdot\phi_C^{(i)}(\beta_i) \\
e_q^{(i)} &\coloneqq \psi_q^{(i-1)}(\beta_i) + \rho_i\cdot\phi_q^{(i)}(\beta_i)
\end{aligned}$$

By examining the equations above, one may see that if an honest prover sent the correct

polynomials then the equations would reduce to

$$\begin{aligned}
e_A^{(i)} &= \widehat{f_A}\big(v_A'^{(i-1)}, ...\gamma^{(i)}(\beta_i)\big) + \rho_i \cdot \widehat{f_A}\big(v_A'^{(i)}, ...\gamma^{(i)}(\beta_i)\big) \\
&= \widehat{f_A}\big(v_A'^{(i-1)}, ...r^{(i)}\big) + \rho_i \cdot \widehat{f_A}\big(v_A'^{(i)}, ...r^{(i)}\big) \\
e_B^{(i)} &= \widehat{f_B}\big(v_B'^{(i-1)}, ...\gamma^{(i)}(\beta_i)\big) + \rho_i \cdot \widehat{f_B}\big(v_B'^{(i)}, ...\gamma^{(i)}(\beta_i)\big) \\
&= \widehat{f_B}\big(v_B'^{(i-1)}, ...r^{(i)}\big) + \rho_i \cdot \widehat{f_B}\big(v_B'^{(i)}, ...r^{(i)}\big) \\
e_C^{(i)} &= \widehat{f_C}\big(v_C'^{(i-1)}, ...\gamma^{(i)}(\beta_i)\big) + \rho_i \cdot \widehat{f_C}\big(v_C'^{(i)}, ...\gamma^{(i)}(\beta_i)\big) \\
&= \widehat{f_C}\big(v_C'^{(i-1)}, ...r^{(i)}\big) + \rho_i \cdot \widehat{f_C}\big(v_C'^{(i)}, ...r^{(i)}\big) \\
e_q^{(i)} &= \widehat{q'^{(i-1)}}\big(...\gamma^{(i)}(\beta_i)\big) + \rho_i \cdot \widehat{q^{(i)}}\big(...\gamma^{(i)}(\beta_i)\big) \\
&= \widehat{q'^{(i-1)}}\big(...r^{(i)}\big) + \rho_i \cdot \widehat{q^{(i)}}\big(...r^{(i)}\big)
\end{aligned}$$

Thus the random combination involving ρ_i is a random combination of the induced polynomials and the quotient polynomials.

Lastly the verifier also applies ρ_i to the commits which encode the witnesses, and to the statements. This is because the random combination of the induced polynomials and quotient polynomials implicitly involves taking the random combination of the vectors ...

$$\begin{aligned}
s'^{(i)} &\coloneqq s'^{(i-1)} + \rho_i \cdot s^{(i)} \in \mathbb{Z}_p \\
c'^{(i)} &\coloneqq \big(c'^{(i-1)}\big)\big(c^{(i)}\big)^{\rho_i} \in \mathbb{G}
\end{aligned} \tag{4.5}$$

The communication complexity from the verifier to the prover, and also the randomness complexity, is two field elements β_i and ρ_i.

Now both the prover and the verifier have the common data necessary for the next phase $i+1$.

$$\mathbb{cd}^{(i+1)} \coloneqq (s'^{(i)}, c'^{(i)}, r^{(i)}, (e_A^{(i)}, e_B^{(i)}, e_C^{(i)}, e_q^{(i)}))$$

As for the pseudo witness vector $w'^{(i+1)}$ necessary for $\mathbb{pd}^{(i+1)}$ the prover may compute it as

$$w'^{(i)} \coloneqq w'^{(i-1)} + \rho_i \cdot w^{(i)} \tag{4.6}$$

4.2.4 Final phase

The final phase protocol **FinalPhase**$(\mathbb{ap}, \mathbb{cd}^{(n+1)}, w'^{(n)})$ is phase $n+1$ and consists of a message from the prover to the verifier followed by the verifier checking the message and then outputting a final decision on acceptance versus rejection regarding all proofs in the batch. Notice how the input to this protocol involves the common data as with the intermediate phases, but the only prover data is $w'^{(n)}$ and not a new statement witness pair because this is the final phase and all statement-witness pairs have been dealt with in the intermediate phases. The common data is as follows.

$$\mathbb{cd}^{(n+1)} = (s'^{(n)}, c'^{(n)}, r^{(n)}, (e_A^{(n)}, e_B^{(n)}, e_C^{(n)}, e_q^{(n)}))$$

The prover and verifier engage as follows.

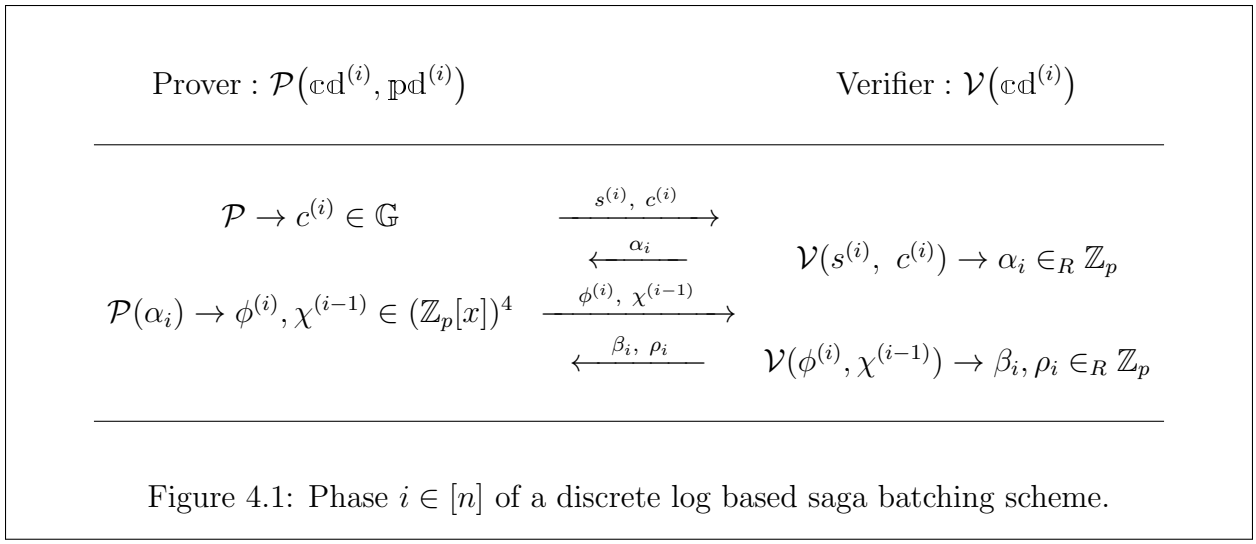

Figure 4.1: Phase $i \in [n]$ of a discrete log based saga batching scheme.

1. The prover opens the commitment $c'^{(n)}$ by sending the verifier the opening vector $(b'', w'^{(n)})$ where one can check that b'' is $b' + \sum_{i\in[n]} b_i\rho_i$.

2. The verifier checks that $(b'', w'^{(n)})$ indeed opens the commitment $c'^{(n)}$ by verifying that the following equality holds.

$$c'^{(n)} = g_{\texttt{Wsize}+1}^{b''} \prod_{j\in[\texttt{Wsize}]} g_j^{w_j'^{(n)}}$$

3. The verifier then parses the pseudo statement vector $s'^{(n)}$ together with the pseudo witness vector $w'^{(n)}$ into pseudo variable vectors $v_A'^{(n)}, v_V'^{(n)}, v_C'^{(n)} \in \mathbb{Z}_p^{\texttt{Vsize}}$ and pseudo quotient polynomial $q'^{(n)} \in \mathbb{Z}_p[x_1, \ldots, x_\ell]$.

   The verifier performs evaluates the induced polynomials and the quotient polynomial at $r^{(n)}$ and accepts if and only if the following equalities hold.

$$\begin{aligned}
\widehat{f_A}\big(v_A'^{(n)}, \ldots r^{(n)}\big) &= e_A^{(n)} \\
\widehat{f_A}\big(v_B'^{(n)}, \ldots r^{(n)}\big) &= e_B^{(n)} \\
\widehat{f_A}\big(v_C'^{(n)}, \ldots r^{(n)}\big) &= e_C^{(n)} \\
\widehat{q'^{(n)}}(\ldots r^{(n)}) &= e_q^{(n)}
\end{aligned}$$

4. The verifier sends its decision to the prover in the form of a bit, 0 for rejection of all n proofs, 1 for acceptance of all n proofs.

This final phase completes the entire saga batching protocol.

4.3 Protocol properties

Theorem 5 (Construction 4.2 is a saga batching scheme)**.** *The construction presented in section 4.2 is a valid saga batching scheme. Let the prime p defining the field $\mathbb{Z}_p$ be chosen such that $p = \text{poly}(2^\lambda)$ for security parameter λ. Let the mask size be* `maskSize` $= 2\ell + 1$.

Completeness: *If a prover is honest and follows the protocol, only trying to prove true statements with valid witnesses, the verifier accepts all proofs in the batch with probability* 1.

Soundness: *If a prover is dishonest, does not follow the protocol or tries to prove false statements, the verifier rejects all proofs in the batch with probability at least* $\text{poly}(\lambda)/p$.

Zero-knowledge: *The prover leaks zero knowledge to the verifier. In particular, the view of the entire interaction from the perspective of the verifier is simulatable. Specifically, the view generated by the simulator is perfectly indistinguishable from the uniform distribution. In other words, the messages sent from the prover appear maximially random to the verifier, the only condition being they pass the verifier's final test.*

Furthermore, since our simulator operates by invoking the verifier as a black-box, we may say our proof is zero-knowledge even when the verifier holds an auxiliary input. By satisfying auxiliary-input zero-knowledge our proof may be employed as a subroutine inside larger protocols.

Proof of knowledge: *When the verifier performs randomization on commitments, reducing the number of commitments the verifier must check, the following proof of knowledge property holds. If the prover knows an opening for every output commitment of the randomization, then the prover knows an opening for every input commitment to the randomization. Furthermore, the prover can extract openings to the input commitments from the openings of the output commitments.*

Space complexity: *Examining the construction, one may observe the following space complexity parameters.*

1. *The* Initial communication size *consists of the group vector defining the Pedersen commitment scheme, which is of size* $\log(p)(\texttt{witnessSize} + 1)$.
2. *The* Per proof size *is* 1 *group element and* `statementSize` $+ (4\ell + 3) + (4\ell - 1)$ *field elements.*
3. *The* Final communication size *is* `witnessSize` $+ 1$ *field elements.*

4.3.1 Completeness

One may inspect the construction to see that completeness holds. We clarify two requirements the prover must satisfy in order to complete the proof and state that if the prover follows the protocol these requirements will be met.

The first requirement we clarify is that the prover can open the commitment $c'^{(n)}$ in the final phase. Expanding the definition of $c'^{(n)}$ we see it equals a linear combination of all the other commitments the prover previously sent.

$$c'^{(n)} = c'^{(0)} \cdot \prod_{i \in [n]} \left(c^{(i)}\right)^{\rho_i}$$

Therefore if the prover indeed has openings for all prior commitments then by taking the same combination of the openings the prover may arrive at an opening for $c'^{(n)}$.

The second requirement we clarify is in regard to how phases $i \in [n]$ begin with a set of pending conditions which the verifier must accept in order to accept all prior proofs. In addition there are new conditions the verifier must accept regarding the new statement $s^{(i)}$ together with its commitment $c^{(i)}$. The verifier interacts with the prover in two rounds in each phase $i \in [n]$ and at the end the pending set of requirements along with the new set of requirements are amortized and reduced to a single set of final requirements that become the requirements examined in the next phase. We clarify why the prover is able to satisfy the final set of requirements if it satisfies the pending and new set of requirements. The final set of requirements is a set of evaluations on the induced polynomials and a quotient polynomial. The evaluations the verifier will expect are $e_A^{(i)}, e_B^{(i)}, e_C^{(i)}, e_q^{(i)}$. If the prover follows the protocol the following equality will hold, and thus the prover will be able to satisfy the evaluations via the combination of witness vectors defined in equation 4.6. We show the specific case of $e_A^{(i)}$ but one show the same for $e_B^{(i)}$, $e_C^{(i)}$, or $e_q^{(i)}$.

$$\begin{aligned}
e_A^{(i)} &= \psi_A^{(i-1)}(\beta_i) + \rho_i \cdot \phi_A^{(i)}(\beta_i) \\
&= \beta_i \cdot \chi_A^{(i-1)}(\beta_i) + e_A^{(i-1)} + \rho_i \cdot \widehat{f_A}\left(v_A^{(i)}, \ldots \gamma^{(i)}(\beta_i)\right) \\
&= \widehat{f_A}\left(v_A'^{(i-1)}, \ldots \gamma^{(i)}(\beta_i)\right) + \rho_i \cdot \widehat{f_A}\left(v_A^{(i)}, \ldots \gamma^{(i)}(\beta_i)\right) \\
&= \widehat{f_A}\left(v_A'^{(i-1)}, \ldots r^{(i)}\right) + \rho_i \cdot \widehat{f_A}\left(v_A^{(i)}, \ldots r^{(i)}\right) \\
&= \widehat{f_A}\left(v_A'^{(i-1)} + \rho_i \cdot v_A^{(i)}, \ldots r^{(i)}\right)
\end{aligned}$$

To see that the last equation holds, one may consult the definition of the induced polynomials contained in definition 14. One will see that the induced polynomials are defined as a linear function of the variable vectors. By taking the random combination of the witness vectors $w'^{(i-1)}$ and $w^{(i)}$ as in equation 4.6 one is implicitly taking the same combination of the variable vectors. Therefore if the prover follows the protocol, and both the pending conditions and the new conditions regarding the statement $s^{(i)}$ and the commitment $c^{(i)}$ are met, the final derived conditions passed to the next phase will also hold. Thus completeness is satisfied.

4.3.2 Soundness

We intend to prove that during phase $i \in [n]$ the verifier reduces the task of verifying the new statement $s^{(i)}$ and the existing polynomial evaluations regarding previous proofs, to a new set of polynomial evaluations. Recall that at the beginning of phase $i \in [n]$ the common data is

$$\mathbb{cd}^{(i)} = (s'^{(i-1)}, c'^{(i-1)}, r^{(i-1)}, (e_A^{(i-1)}, e_B^{(i-1)}, e_C^{(i-1)}, e_q^{(i-1)}))$$

The pending conditions on this data conditioning whether or not the verifier should accept the previous $i-1$ proofs, is the following. Commitment $c'^{(i-1)}$ should contain a pseudo witness vector $w'^{(i-1)}$ such that the following equalities hold with respect to the pseudo variable vectors $v_A'^{(i-1)}, v_B'^{(i-1)}, v_C'^{(i-1)}$ derived from $s'^{(i-1)}$ and $w'^{(i-1)}$ as well as the pseudo quotient polynomial $q'^{(i-1)}$.

$$\begin{aligned}\widehat{f_A}(v_A'^{(i-1)}, \ldots r^{(i-1)}) &= e_A^{(i-1)}\\ \widehat{f_B}(v_B'^{(i-1)}, \ldots r^{(i-1)}) &= e_B^{(i-1)}\\ \widehat{f_A}(v_C'^{(i-1)}, \ldots r^{(i-1)}) &= e_C^{(i-1)}\\ \widehat{q'^{(i-1)}}(\ldots r^{(i-1)}) &= e_q^{(i-1)}\end{aligned}$$

With the introduction of a new statement $s^{(i)}$ with commitment $c^{(i)}$ new conditions must be satisfied in addition to the pending conditions, and both set of conditions will be amortized into a final set of conditions passed to the next phase. We state now that the final pending conditions appearing at the start of phase $n+1$ are verified directly by the verifier as seen in the construction. Therefore we are left to prove that in each of the intermediate phases the pending conditions together with the new conditions are appropriately amortized into pending conditions for the next phase.

First we focus on reducing the validity of $s^{(i)}$ together with the witness $w^{(i)}$ encoded in $c^{(i)}$ to verifying evaluations on the quotient polynomial $q^{(i)}$ and the three polynomials induced by the three variable vectors derived from $s^{(i)}$ and $w^{(i)}$. Suppose the four ϕ polynomials sent by the prover in step 4 are correct. Then we have

$$\begin{aligned}\phi_A^{(i)}(x) &= \widehat{f_A}\big(v_A^{(i)}, \ldots \gamma^{(i)}(x)\big)\\ \phi_B^{(i)}(x) &= \widehat{f_B}\big(v_B^{(i)}, \ldots \gamma^{(i)}(x)\big)\\ \phi_C^{(i)}(x) &= \widehat{f_C}\big(v_C^{(i)}, \ldots \gamma^{(i)}(x)\big)\\ \phi_q^{(i)}(x) &= \widehat{q^{(i)}}\big(\ldots \gamma^{(i)}(x)\big)\end{aligned}$$

In step 5 the verifier checks that

$$\phi_A^{(i)}(1) \cdot \phi_B^{(i)}(1) + \phi_C^{(i)}(1) = \phi_q^{(i)}(1) / \big(\alpha_i^{\texttt{Ccount}} - 1\big)$$

By substitution and reduction we see that this equality may be translated to

$$\begin{aligned}\widehat{f_A}\big(v_A^{(i)}, \ldots [\alpha_i^{2^{j-1}}]_{j\in[\ell]}\big) \cdot \widehat{f_B}\big(v_B^{(i)}, \ldots [\alpha_i^{2^{j-1}}]_{j\in[\ell]}\big) + \widehat{f_C}\big(v_C^{(i)}, \ldots [\alpha_i^{2^{j-1}}]_{j\in[\ell]}\big)\\ = \widehat{q^{(i)}}\big(\ldots [\alpha_i^{2^{j-1}}]_{j\in[\ell]}\big) / \big(\alpha_i^{\texttt{Ccount}} - 1\big)\end{aligned}$$

However, by definition 15 we know this further reduces to

$$f_A\big(v_A^{(i)}, \alpha_i\big) \cdot f_B\big(v_B^{(i)}, \alpha_i\big) + f_C\big(v_C^{(i)}, \alpha_i\big) = q^{(i)}\big(\alpha_i\big)\big(\alpha_i^{\texttt{Ccount}} - 1\big)$$

If the equality does not hold, the verifier will immediately reject. Therefore we may assume the equality holds. By the Schwartz-Zippel lemma, the verifier concludes that with

high probability, in particular probability at least $1 - 2\texttt{Ccount}/p$, the following polynomial equation holds.

$$f_A\big(v_A^{(i)}, x\big) \cdot f_B\big(v_B^{(i)}, x\big) + f_C\big(v_C^{(i)}, x\big) = q^{(i)}\big(x\big)\big(x^{\texttt{Ccount}} - 1\big)$$

By proposition 4 we may replace $x^{\texttt{Ccount}} - 1$ with $\prod_{h \in H}(x - h)$. Therefore the left hand side vanishes on all $h \in H$. By definition 14 of arithmetic program satisfaction, the verifier may conclude that $w^{(i)}$ is a valid witness for statement $s^{(i)}$ as long as the four ϕ polynomials are correct.

Suppose any of the four ϕ polynomials is incorrect. We consider the case of $\phi_A^{(i)}$ but one may say analogous statements for $\phi_B^{(i)}$, $\phi_C^{(i)}$, and $\phi_q^{(i)}$. To say $\phi_A^{(i)}$ is incorrect we mean that it does not satisfy the definitional equality shown in 4.2. To detect such an inequality, the verifier uses the Schwartz-Zippel lemma and chooses a random $\beta_i \in \mathbb{Z}_p$ for identity testing. With high probability, this time with probability at least $1 - (\ell + 1)/p$, we will have the following inequality.

$$\phi_A^{(i)}(\beta_i) \neq \widehat{f_A}\big(v_A^{(i)}, ...\gamma^{(i)}(\beta_i)\big) = \widehat{f_A}\big(v_A^{(i)}, ...r^{(i)}\big)$$

where the reduction to $r^{(i)}$ comes from definition 4.3. We will come back to these inequalities after returning to analysis of the pending conditions.

Now suppose any of the pending conditions are incorrect. Then at least one of the four χ polynomials sent by the prover in step 4 is incorrect. To see this, suppose for sake of contradiction all four χ polynomials are correct. Then by definition 4.4 of the ψ polynomials upon substituting in the correct χ polynomials and then plugging in 0 we get

$$\begin{aligned}
\psi_A^{(i-1)}(0) &= \widehat{f_A}\big(v_A'^{(i-1)}, ...r^{(i-1)}\big) \\
\psi_B^{(i-1)}(0) &= \widehat{f_B}\big(v_B'^{(i-1)}, ...r^{(i-1)}\big) \\
\psi_C^{(i-1)}(0) &= \widehat{f_C}\big(v_C'^{(i-1)}, ...r^{(i-1)}\big) \\
\psi_q^{(i-1)}(0) &= \widehat{q^{(i-1)}}\big(...r^{(i-1)}\big)
\end{aligned}$$

where the reduction to $r^{(i-1)}$ comes from the constraint $\gamma^{(i)}(0) = r^{(i-1)}$. On the other hand, instead of expanding the definition of the ψ polynomials, suppose we directly plug in 0.

$$\begin{aligned}
\psi_A^{(i-1)}(0) &= 0 \cdot \chi_A^{(i-1)}(x) + e_A^{(i-1)} = e_A^{(i-1)} \\
\psi_B^{(i-1)}(0) &= 0 \cdot \chi_B^{(i-1)}(x) + e_B^{(i-1)} = e_B^{(i-1)} \\
\psi_C^{(i-1)}(0) &= 0 \cdot \chi_C^{(i-1)}(x) + e_C^{(i-1)} = e_C^{(i-1)} \\
\psi_q^{(i-1)}(0) &= 0 \cdot \chi_q^{(i-1)}(x) + e_q^{(i-1)} = e_q^{(i-1)}
\end{aligned}$$

Using the transitivity of equality in the above two sets of equations we see that the pending conditions are all satisfied, thus a contradiction.

Therefore we may conclude at least one of the χ polynomials is incorrect, in which case by expanding the definition 4.4 of the ψ polynomials we will have at least one of the following

polynomial inequalities.

$$\psi_A^{(i-1)}(x) \neq \widehat{f_A}\big(v_A'^{(i-1)}, ...\gamma^{(i)}(x)\big)$$
$$\psi_B^{(i-1)}(x) \neq \widehat{f_B}\big(v_B'^{(i-1)}, ...\gamma^{(i)}(x)\big)$$
$$\psi_C^{(i-1)}(x) \neq \widehat{f_C}\big(v_C'^{(i-1)}, ...\gamma^{(i)}(x)\big)$$
$$\psi_q^{(i-1)}(x) \neq \widehat{q'^{(i-1)}}\big(...\gamma^{(i)}(x)\big)$$

To detect such an inequality, the verifier uses the Schwartz-Zippel lemma and chooses a random $\beta_i \in \mathbb{Z}_p$ for identity testing. We consider the case of $\psi_A^{(i)}$ but one may say analogous statements for $\psi_B^{(i)}$, $\psi_C^{(i)}$, and $\psi_q^{(i)}$. With high probability, with probability at least $1-(\ell+1)/p$, we will have the following inequality.

$$\psi_A^{(i-1)}(\beta_i) \neq \widehat{f_A}\big(v_A^{(i-1)}, ...\gamma^{(i)}(\beta_i)\big) = \widehat{f_A}\big(v_A^{(i-1)}, ...r^{(i)}\big)$$

where the reduction to $r^{(i)}$ comes from definition 4.3.

At this point we have reduced to detecting any of eight possible inequalities, four with respect to evaluating the ϕ polynomials on β_i and four with respect to evaluating the ψ polynomials on β_i. These two sets of evaluation checks may be reduced to one by a random linear combination, due to the linearity of polynomial evaluations with respect to the same inputs. The random scalar for the linear combination is ρ_i and the new evaluations are computed as

$$e_A^{(i)} = \psi_A^{(i-1)}(\beta_i) + \rho_i \cdot \phi_A^{(i)}(\beta_i) = \widehat{f_A}\big(v_A'^{(i-1)}, ...r^{(i)}\big) + \rho_i \cdot \widehat{f_A}\big(v_A'^{(i)}, ...r^{(i)}\big)$$
$$e_B^{(i)} = \psi_B^{(i-1)}(\beta_i) + \rho_i \cdot \phi_B^{(i)}(\beta_i) = \widehat{f_B}\big(v_B'^{(i-1)}, ...r^{(i)}\big) + \rho_i \cdot \widehat{f_B}\big(v_B'^{(i)}, ...r^{(i)}\big)$$
$$e_C^{(i)} = \psi_C^{(i-1)}(\beta_i) + \rho_i \cdot \phi_C^{(i)}(\beta_i) = \widehat{f_C}\big(v_C'^{(i-1)}, ...r^{(i)}\big) + \rho_i \cdot \widehat{f_C}\big(v_C'^{(i)}, ...r^{(i)}\big)$$
$$e_q^{(i)} = \psi_q^{(i-1)}(\beta_i) + \rho_i \cdot \phi_q^{(i)}(\beta_i) = \widehat{q'^{(i-1)}}\big(...r^{(i)}\big) + \rho_i \cdot \widehat{q^{(i)}}\big(...r^{(i)}\big)$$

In step 5 the verifier takes the same random combination of the commitments, see equation 4.5. Doing so implicitly take the same combination of the encoded witnesses, see equation 4.6. The combination of witnesses results in pseudo vectors $v_A'^{(i)}, v_A'^{(i)}, v_A'^{(i)}$ and pseudo polynomial $q'^{(i)}$ such that both sets of evaluations have been reduced to the following set of evaluations.

$$e_A^{(i)} = \widehat{f_A}\big(v_A'^{(i)}, ...r^{(i)}\big)$$
$$e_B^{(i)} = \widehat{f_B}\big(v_B'^{(i)}, ...r^{(i)}\big)$$
$$e_C^{(i)} = \widehat{f_C}\big(v_C'^{(i)}, ...r^{(i)}\big)$$
$$e_q^{(i)} = \widehat{q'^{(i)}}\big(...r^{(i)}\big)$$

Therefore in round $i \in [n]$ the verifier successfully reduces the pending conditions along with the new conditions regarding the new statement $s^{(i)}$ and commitment $c^{(i)}$ to a new set of pending conditions.

4.3.3 Zero-knowledge

The simulator $\mathcal{S}$ operates as follows.

Setup:

The simulator uniformly selects a vectors $u \in_R \mathbb{Z}_p^{(\texttt{Wsize}+1)}$ with the condition that $u_{(\texttt{Wsize}+1)} \neq 0$. Then the simulator generates the vector $g \in \mathbb{G}^{(\texttt{Wsize}+1)}$ defining the Pedersen commitment scheme as follows. Let $\zeta \in \mathbb{G}$ be a generator of $\mathbb{G}$. Since $\mathbb{G}$ is cyclic and of prime order any non-identity element of $\mathbb{G}$ may function as the generator ζ. Then $[g_i]_{i\in(\texttt{Wsize}+1)]} = [\zeta^{u_i}]_{i\in(\texttt{Wsize}+1)]}$. In other words, cryptographic constant g_i is the generator ζ raised to the power u_i.

Initial phase (0):

In the initial phase, that is phase 0, the adversary chooses uniform commitment mask $b' \in_R \mathbb{Z}_p$ and evaluations $e_A^{(0)}, e_B^{(0)}, e_C^{(0)}, e_q^{(0)} \in_R \mathbb{Z}_p$ uniformly. From b' the simulator computes the commitment $c'^{(0)}$ as

$$c'^{(0)} \coloneqq g_{(\texttt{Wsize}+1)}^{b'}$$

The simulator sends these values to the verifier, updating the verifier's state, but does not expect a return value from the verifier until phases $i \in [n]$. Thus the simulator invokes the verifier as such

$$\mathcal{V}(c'^{(0)}, e_A^{(0)}, e_B^{(0)}, e_C^{(0)}, e_q^{(0)})$$

Steps 1 and 2 of intermediate phase $i \in [n]$:

The simulator selects any statement it wishes to simulate proving in phase i as $s^{(i)} \in \mathbb{Z}_p^{\texttt{Ssize}}$. For the witness for statement $s^{(i)}$ the simulator uses an arbitrary witness vector, and for simplicity we will use the zero vector $[0]_{i\in\texttt{Wsize}}$. Therefore the simulator need not commitment to the witness vector, but only to a commitment mask. The simulator uniformly chooses a commitment mask $b_i \in \mathbb{Z}_p$ and computes the commitment as

$$c^{(i)} \coloneqq g_{(\texttt{Wsize}+1)}^{b_i}$$

Then the simulator invokes the verifier on statement $s^{(i)}$ and commitment $c^{(i)}$, receiving back the challenge α_i.

$$\mathcal{V}\big(s^{(i)}, c^{(i)}\big) \to \alpha_i$$

Step 3 of intermediate phase $i \in [n]$:

The simulator computes the line $\gamma^{(i)} = (\gamma_j^{(i)} \in \mathbb{Z}_p[x])_{j\in[\ell]}$ as

$$\gamma_j^{(i)}(x) = (1-x)r_j^{(i-1)} + x \cdot \alpha_i^{j-1}$$

Steps 4 and 5 of intermediate phase $i \in [n]$:

The simulator selects the four ϕ univariate polynomials uniformly conditioned such that equation 4.1 will be satisfied.

$$\begin{aligned} \phi_A^{(i)}, \phi_B^{(i)}, \phi_C^{(i)} &\in_R \mathbb{Z}_p^{<\ell+1}[x] \\ \phi_q^{(i)} &\in_R \mathbb{Z}_p^{<\ell}[x] \\ \text{s.t. } \phi_A^{(i)}(1) \cdot \phi_B^{(i)}(1) + \phi_C^{(i)}(1) &= \phi_q^{(i)}(1) \end{aligned}$$

To do so the simulator may choose the first three polynomials uniformly and constrain a single coefficient of $\phi_q^{(i)}$. The four χ univariate polynomials are sample uniformly with no conditioning

$$\begin{aligned} \chi_A^{(i-1)}, \chi_B^{(i-1)}, \chi_C^{(i-1)} &\in_R \mathbb{Z}_p^{<\ell}[x] \\ \chi_q^{(i-1)} &\in_R \mathbb{Z}_p^{<\ell-1}[x] \end{aligned}$$

Feeding the eight polynomials to the verifier the simulator receives β_i and ρ_i in return.

$$\mathcal{V}\big(\phi_A^{(i)}, \phi_B^{(i)}, \phi_C^{(i)}, \phi_q^{(i)}, \chi_A^{(i-1)}, \chi_B^{(i-1)}, \chi_C^{(i-1)}, \chi_q^{(i-1)}\big) \to \beta_i,\ \rho_i$$

The simulator discards β_i and ρ_i except in phase n in which case the simulator carries them over to the final phase.

Final phase $(n+1)$:

Using β_n from phase n the simulation computes $r^{(n)}$ as

$$r^{(n)} = \Big[r_j^{(n)}\Big]_{j\in[\ell]} = \Big[\gamma_j^{(n)}(\beta^{(n)})\Big]_{j\in[\ell]} \in \mathbb{Z}_p^{\ell}$$

Now the common data held by both the simulator and the verifier is

$$\mathbb{cd}^{(n+1)} = (s'^{(n)}, c'^{(n)}, r^{(n)}, (e_A^{(n)}, e_B^{(n)}, e_C^{(n)}, e_q^{(n)}))$$

The simulator is tasked with sending the verifier a vector $(b'', w'^{(n)}) \in \mathbb{Z}_p^{\texttt{Wsize}+1}$ that opens via the Pedersen commitment scheme to $c'^{(n)}$. Furthermore, the simulator must choose $w'^{(n)}$ such that the corresponding variable vectors $v'^{(n)}_A, v'^{(n)}_B, v'^{(n)}_B$ and quotient polynomial $q'^{(n)}$ satisfy the following equations.

$$\widehat{f_A}\big(v_A'^{(n)}, \ldots r^{(n)}\big) = e_A^{(n)} \tag{4.7}$$
$$\widehat{f_B}\big(v_B'^{(n)}, \ldots r^{(n)}\big) = e_B^{(n)} \tag{4.8}$$
$$\widehat{f_C}\big(v_C'^{(n)}, \ldots r^{(n)}\big) = e_C^{(n)} \tag{4.9}$$
$$\widehat{q'^{(n)}}(\ldots r^{(n)}) = e_q^{(n)} \tag{4.10}$$

To satisfy both these equation and the opening, the simulator strategically chooses b'' and $w'^{(n)}$.

First, the simulator choose $w'^{(n)} = (t'^{(n)}, (m_A'^{(n)}, m_B'^{(n)}, m_C'^{(n)}), q'^{(n)}) \in \mathbb{Z}_p^{\texttt{Wsize}}$ by selecting $t' \in \mathbb{Z}_p^{\texttt{Tsize}}$ uniformly. Then the simulator selects $(m'_A)_i^{(n)}, (m'_B)_i^{(n)}, (m'_C)_i^{(n)} \in_R \mathbb{Z}_p$ uniformly for $i \in \{2, \ldots, \texttt{Msize}\}$. In order for the variable vectors $v'^{(n)}_A, v'^{(n)}_B, v'^{(n)}_B$ to satisfy the first three equations above the simulator chooses $(m_A)_1^{(n)}, (m_B)_1^{(n)}, (m_C)_1^{(n)} \in \mathbb{Z}_p$ stategically as follows. Recall that by definitions of subsets $H'_A, H'_B, H'_C \subseteq H$, values $(m_A)_1^{(n)}, (m_B)_1^{(n)}, (m_C)_1^{(n)}$ effectively determine the evaluation of univariate polynomials $f_A(v_A'^{(n)}, x), f_B(v_B'^{(n)}, x), f_C(v_C'^{(n)}, x)$ each at certain points $h_a \in H'_A$, $h_b \in H'_B$, and $h_c \in H'_C$ repsecitlvey. This lends one degree of freedom to each of the univariate poynomials which means exactly one degree of freedom to each of the multivariate versions of the polynomials $\widehat{f_A}(v_A'^{(n)}, \ldots x), \widehat{f_B}(v_B'^{(n)}, \ldots x), \widehat{f_C}(v_C'^{(n)}, \ldots x)$. The simulator may use these degrees of freedom to formulate the multivariate polynomials such that they each satisfy the single constraints shown in equation 4.7, 4.8, and 4.9. Thus the simulator explicitly determines $(m_A)_1^{(n)}, (m_B)_1^{(n)}, (m_C)_1^{(n)}$ via standard polynomial interpolation techniques. We mention that while the single degrees of freedom are sufficient, many others also are available via the other indices of the masks $(m_A)^{(n)}, (m_B)^{(n)}, (m_C)^{(n)}$.

Likewise, to make sure equation 4.10 is satisfied the simulator chooses $q'^{(n)}$ strategically. Coefficients $q_i \in_R \mathbb{Z}_p$ for $i \in \{2, \ldots, \texttt{Qsize}\}$ are chosen uniformly. The constant coefficient (or any other could have been chosen) q_1 is left as the single degree of freedom such that equation 4.10 holds.

Now that the pseudo witness vector $w'^{(n)}$ is set, the simulator must make sure it opens to the commitment $c'^{(n)}$. This is done by appropriately choosing the commitment mask $b'' \in \mathbb{Z}_p$ for the commitment of $w'^{(n)}$. According to equation 4.5, upon unrolling the definition of $c'^{(n)}$ we get

$$c'^{(n)} = c'^{(0)} \prod_{i \in [n]} \left(c^{(i)}\right)^{\rho_i}$$

Recall that the simulator set $c'^{(0)}$ and $c^{(i)}$ for $i \in [n]$ according to the following defintions.

$$c'^{(0)} = g^{b'}_{(\texttt{Wsize}+1)} \text{ and } \forall i \in [n],\ c^{(i)} = g^{b_i}_{(\texttt{Wsize}+1)}$$

Interpreting the former equation using the latter equalities for substitution we get

$$\begin{aligned}
c'^{(n)} &= c'^{(0)} \prod_{i \in [n]} \left(c^{(i)}\right)^{\rho_i} \\
&= g^{b'}_{(\texttt{Wsize}+1)} \prod_{i \in [n]} g^{b_i \rho_i}_{(\texttt{Wsize}+1)} \\
&= \zeta^{u_{(\texttt{Wsize}+1)} b'} \prod_{i \in [n]} \zeta^{u_{(\texttt{Wsize}+1)} b_i \rho_i} \\
&= \exp\left(\zeta, u_{(\texttt{Wsize}+1)} b' + \sum_{i \in [n]} u_{(\texttt{Wsize}+1)} b_i \rho_i\right) \\
&= \exp\left(\zeta, u_{(\texttt{Wsize}+1)} \left(b' + \sum_{i \in [n]} b_i \rho_i\right)\right)
\end{aligned}$$

On the other hand, having defined $w'^{(n)}$ we need $c'^{(n)}$ to open to b'' and $w'^{(n)}$ by satisfaction

of the following equation.

$$\begin{aligned} c'^{(n)} &= g_{(\texttt{Wsize}+1)}^{b''} \prod_{j \in [\texttt{Wsize}]} g_j^{w'^{(n)}_j} \\ &= \zeta^{u_{(\texttt{Wsize}+1)} b''} \prod_{j \in [\texttt{Wsize}]} \zeta^{u_j w'^{(n)}_j} \\ &= \exp\left(\zeta, u_{(\texttt{Wsize}+1)} b'' + \sum_{j \in [\texttt{Wsize}]} u_j w'^{(n)}_j\right) \end{aligned}$$

Therefore we set b'' to the unique value that satisfies the following field equation, which exists because $u_{(\texttt{Wsize}+1)} \neq 0$.

$$u_{(\texttt{Wsize}+1)}\left(b' + \sum_{i \in [n]} b_i \rho_i\right) = u_{(\texttt{Wsize}+1)} b'' + \sum_{j \in [\texttt{Wsize}]} u_j w'^{(n)}_j$$

Finally the simulator sends the vector $(b'', w'^{(n)}_j)$ to the verifier. This completes the description of the simulator.

The simulator constructed above clearly outputs a view that is maximally uniform apart from the necessary conditions for convincing the verifier, namely constraining the four ϕ polynomials with one degree of freeom such that they satisfy the quadratic equation 4.1 when invoked on input 1, and constraining b'' in the last message $(b'', w'^{(n)})$ such that the vector opens the commitment $c^{(n)}$. This is the most uniform distribution possible constrained on the simulator convincing the verifier of all statements $s^{(i)}$ for $i \in [n]$. We are therefore left to prove that the view from an honest verifier would also be maximally uniform, thus equating the simulator's view distribution to the honest prover's view distribution fulfilling the requirement of zero-knowledge. We prove that the honest prover's view is maximally uniform by proving that all messages sent by the prover are uniform except for the last four ϕ polynomials which are again constrained for sake of the quadratic equation, and the last message which is again constrained on being a valid opening to commitment $c^{(n)}$.

In the first phase, that is 0, the prover sends commitment to uniform vector $w'^{(0)}$. The commitment to $w'^{(0)}$ is uniform by the uniformity of the commitment mask b'. Likewise, all n subsequent commitments sent by the prover are uniform in the group $\mathbb{G}$ due to the uniformy of the commitment masks b_i for $i \in [n]$. In the last phase, that is phase $n+1$, the prover sends the vector $(b'', w'^{(n)})$ where $w'^{(n)}$ can be unrolled by definition 4.6 and seen to equal

$$w'^{(n)} = w'^{(0)} + \sum_{i \in [n]} \rho_i \cdot w^{(i)}$$

Since $w'^{(0)}$ is selected uniformly (in phase 0), the vector $w'^{(n)}$ is also distributed uniformly. Thus all messages sent by the prover to the verifier in phases 0 and $n+1$ are maximally uniform. In each of the intermediate phases $i \in [n]$, in step 1 the prover sends a statment and a uniform commitment. In step 2, 3, and 5 the prover sends nothing. In order to establish that all prover messages through all intermediate phases are uniform, are are left to prove that in step 4 for phases $i \in [n]$ the four ϕ are maximally uniform and the four χ polynomials are fully uniform.

The purpose of the subsets $H'_A, H'_B, H'_C \subseteq H$ defined in the third item in definition 12, is to allow the `maskSize` degrees of freedom contained in each of the three masks $m_A^{(i)}, m_B^{(i)}, m_C^{(i)}$ to be transferred via variable vectors $v_A^{(i)}, v_B^{(i)}, v_C^{(i)}$ to the polynomials

$$\widehat{f_A}\big(v_A^{(i)}, ...x\big),\ \widehat{f_B}\big(v_B^{(i)}, ...x\big),\ \widehat{f_C}\big(v_C^{(i)}, ...x\big)$$

This way each of the three polynomials above has `maskSize` degrees of freedom for randomization. In step 4 of phases $i \in [n]$, the prover will send the three ϕ polynomials $\phi_A^{(i)}(x)$, $\phi_B^{(i)}(x)$, and $\phi_C^{(i)}(x)$ each of degree less than $\ell + 1$. By sending the $\ell + 1$ field elements defining $\phi_A^{(i)}(x)$, the prover reveals at most $\ell + 1$ degrees of freedom regarding the polynomial $\widehat{f_A}\big(v_A^{(i)}, ...x\big)$, and likewise we may say the same for the likes of B and C. The number degrees of freedom in these induced polynomials is equal to the number of degrees of freedom in the variable vectors $v_A^{(i)}, v_B^{(i)}, v_C^{(i)}$.

Analogously, we consider the polynomials

$$\widehat{f_A}\big(v_A^{\prime(i-1)}, ...x\big),\ \widehat{f_B}\big(v_B^{\prime(i-1)}, ...x\big),\ \widehat{f_C}\big(v_C^{\prime(i-1)}, ...x\big)$$

Analogous to the three ϕ polynomials, in step 4 of phases $i \in [n]$ the prover sends the three χ polynomials $\chi_A^{(i-1)}(x), \chi_B^{(i-1)}(x), \chi_B^{(i-1)}(x)$. They are each of degree less than ℓ, thus revealing ℓ of the remaining degrees of freedom contained in $v_A^{\prime(i-1)}, v_B^{\prime(i-1)}, v_C^{\prime(i-1)}$.

Now we put these two observations together to define the number of degrees of freedom remaining in each of $v_A^{\prime(i)}, v_B^{\prime(i)}, v_C^{\prime(i)}$ which we denote by $\texttt{dof}^{(i)}$. Implicitly by definition 4.6, this next generation of pseudo variables is defined as

$$\begin{aligned} v_A^{\prime(i)} &\coloneqq v_A^{\prime(i-1)} + \rho_i \cdot v_A^{(i)} \\ v_B^{\prime(i)} &\coloneqq v_B^{\prime(i-1)} + \rho_i \cdot v_B^{(i)} \\ v_C^{\prime(i)} &\coloneqq v_C^{\prime(i-1)} + \rho_i \cdot v_C^{(i)} \end{aligned}$$

Given this random combination, we may compute $\texttt{dof}^{(i)}$ as the remaining degrees of freedom in each of $v_A^{\prime(i-1)}, v_B^{\prime(i-1)}, v_C^{\prime(i-1)}$ *plus* the remaining degrees of freedom in each of $v_A^{(i)}, v_B^{(i)}, v_C^{(i)}$. By our first observation above, the number of degrees of freedom remaining in each of $v_A^{(i)}, v_B^{(i)}, v_C^{(i)}$ is $\texttt{maskSize} - (\ell + 1)$. By our second observation above, the number of degrees of freedom remaining in each of $v_A^{\prime(i-1)}, v_B^{\prime(i-1)}, v_C^{\prime(i-1)}$ is $\texttt{dof}^{(i-1)} - \ell$. Summing these together we obtain the number of degrees of freedom remaining in each of $v_A^{(i)}, v_B^{(i)}, v_C^{(i)}$ at the beginning of phase i.

$$\texttt{dof}^{(i)} = \big(\texttt{maskSize} - (\ell + 1)\big) + \big(\texttt{dof}^{(i-1)} - \ell\big)$$

Now we note that $\texttt{dof}^{(0)} = \texttt{maskSize}$ because in phase 0 the prover selects each of the three masks contained in $w'^{(0)}$ uniformly. This yields a recurrence sequence. In order for the number of proofs that are executed to be an arbitrary polynomial in the security parameter, that is $n = \text{poly}(\lambda)$, we should keep $\texttt{dof}^{(i)}$ constant. Therefore we must choose `maskSize` such that $\texttt{dof}^{(i-1)} = \texttt{dof}^{(i)}$, and rearranging the equation this yields $\texttt{maskSize} = 2\ell + 1$ as written in theorem 5.

4.3.4 Proof of knowledge

Theorem 6 (Proof of knowledge for Pedersen commitments)**.** *Suppose $\mathbb{G}$ is an abelian group of prime order written in multiplicative notation and $g \in \mathbb{G}^k$ is a vector of constants from the group defining a Pedersen commitment scheme for the field $\mathbb{Z}_p$ for prime p where $|\mathbb{G}| = p$. Suppose an adversary gives two commitments $c_1, c_2 \in \mathbb{G}$ claiming to hold openings for them. Suppose a challenger issues the challenge $\alpha \in \mathbb{Z}_p$ and the adversary is tasked with returning an opening to the commitment $c_1 \cdot c_2^{\alpha}$. If the adversary can succeed with probability at least $\epsilon > 1/2^{\lambda}$ over the selection of α and over the adversary's internal randomness, then there exists an extractor $\mathcal{E}$ such that the adversary may employ $\mathcal{E}$ as a subroutine to extract openings for c_1 and c_2 in expected time* $\mathrm{poly}(\lambda)/\epsilon$.

Proof. By conditioned imposed by soundness analysis on the size of the field relative to the security parameter due to the Schwartz-Zippel lemma, we may assume $2/p \leq 1/2^{\lambda} < \epsilon$. Then by a counting argument, since the adversary succeeds with probability at least ϵ over both the selection of the challenge and the adversary's internal randomness, the adversary must succeed with probability at least ϵ over the selection of the challenge alone. Indeed, consider a truth table with rows labelled by challenges and column labelled by the adversary's coins. By assumption at least an ϵ fraction of entries are true. Therefore at least an ϵ fraction of rows contain at least a single true value.

The extractor $\mathcal{E}$ operates as follows. It issues random challenges until it has found two distinct challenges $\alpha_1, \alpha_2 \in \mathbb{Z}_p$ on which the adversary successfully opens commitments $c_1 \cdot c_2^{\alpha_1}$ and $c_1 \cdot c_2^{\alpha_2}$. We denote these openings by $v^{(1)}, v^{(2)} \in \mathbb{Z}_p^k$ respectively. Using these openings $\mathcal{E}$ computes openings $w^{(1)}, w^{(2)} \in \mathbb{Z}_p^k$ for commitment c_1 and c_2 respectively as follows.

$$
\begin{aligned}
w_i^{(1)} &= (\alpha_2 v_i^{(1)} - \alpha_1 v_i^{(2)})/(\alpha_2 - \alpha_1) \\
w_i^{(2)} &= (v_i^{(1)} - v_i^{(2)})/(\alpha_1 - \alpha_2)
\end{aligned}
$$

These are indeed valid openings because we have

$$
\begin{aligned}
\prod_{i \in [k]} g_i^{w^{(1)}} &= \prod_{i \in [k]} g_i^{\left(\alpha_2 v_i^{(1)} - \alpha_1 v_i^{(2)}\right)/(\alpha_2 - \alpha_1)} \\
&= \left(\frac{(c_1 \cdot c_2^{\alpha_1})^{\alpha_2}}{(c_1 \cdot c_2^{\alpha_2})^{\alpha_1}} \right)^{1/(\alpha_2 - \alpha_1)} = \left(\frac{c_1^{\alpha_2}}{c_1^{\alpha_1}} \right)^{1/(\alpha_2 - \alpha_1)} = c_1
\end{aligned}
$$

$$
\begin{aligned}
\prod_{i \in [k]} g_i^{w^{(2)}} &= \prod_{i \in [k]} g_i^{\left(v_i^{(1)} - v_i^{(2)}\right)/(\alpha_1 - \alpha_2)} \\
&= \left(\frac{c_1 \cdot c_2^{\alpha_1}}{c_1 \cdot c_2^{\alpha_2}} \right)^{1/(\alpha_1 - \alpha_2)} = \left(\frac{c_2^{\alpha_1}}{c_2^{\alpha_2}} \right)^{1/(\alpha_1 - \alpha_2)} = c_2
\end{aligned}
$$

Regarding the running time of $\mathcal{E}$ it requires expected running time $1/\epsilon$ in order to issue a challenge the adversary can answer. Therefore it requires expected running time $\mathcal{O}(\lambda)/\epsilon$ to find two challenges on which the adversary succeeds, and as demonstrated above two successes are sufficient. □

Chapter 5

Saga Batching in the Lattice Setting

5.1 Definitions

Despite having already defined a model for arithmetic programs as defined in section 12, we define a slightly different model below in definition 20. Before doing so, however, we first define several other definitions that will be crucial throughout our lattice based construction. First is the Monomial function 16 which represents the monomials of multilinear polynomials which we will use several times in sumcheck protocol. Second is the Equal function 17, which is the building block of the correspondence between vectors and multilinear polynomials, in particular the notion of the multilinear extension of a vector as defined next in definition 19. Also important to the definition of a multilinear extension, and used frequently throughout our construction, are the binary decomposition and composition functions BD, BC as defined in definition 18.

Definition 16 (The Monomial function)**.** *The function* Monomial_α *for some natural number* $\alpha \in \mathbb{N}$ *has the signature* $\mathbb{Z}_p^\alpha \times \mathbb{Z}_p^\alpha \to \mathbb{Z}_p$*. The purpose of the function is to let the left input use a binary representation to select one of the* 2^α *monomials in a multilinear polynomial should it be invoked on the right input. Therefore the definition is as follows.*

$$\text{Monomial}_\alpha(x, r) \coloneqq \prod_{i=1}^{\alpha} x_i(r_i - 1) + 1$$

This way, r_i *appears in the monomial if and only if* $x_i = 1$*.*

Definition 17 (The Equal function)**.** *The function* Equal_α *for some natural number* $\alpha \in \mathbb{N}$ *has the signature* $\mathbb{Z}_p^\alpha \times \mathbb{Z}_p^\alpha \to \mathbb{Z}_p$*. The purpose of the function is to compare the left and right input in the case that they are both binary inputs, and output* 1 *if they are equal and* 0 *otherwise. Therefore the definition is as follows.*

$$\text{Equal}_\alpha(x, y) \coloneqq \prod_{i=1}^{\alpha} (1 - x_i)(1 - y_i) + x_i \cdot y_i$$

This way the i'th term is 1 *if and only if* $x_i = y_i \in \{0, 1\}$*.*

Definition 18 (The BD and BC functions)**.** *The binary decomposition and composition functions* BD *and* CD *respectively. They are intended to decompose and compose between elements in $\mathbb{Z}_p$ and their binary representations. There is always some implicit range in which the $\mathbb{Z}_p$ values may exist and the binary forms consist of the minimum number of bits to represent that range. By default we may we the entire range $\{0, \ldots, p\}$.*

The decomposition function has the signature $\mathbb{Z}_p \to \{0,1\}^{\lceil \log(p) \rceil}$. The composition function has the signature $\{0,1\}^{\lceil \log(p) \rceil} \to \mathbb{Z}_p$. We define the composition function below and the decomposition function may naturally be defined as its inverse.

$$\mathrm{BC}(\ldots x) \coloneqq \sum_{i=1}^{\lceil \log(p) \rceil} x_i \cdot 2^i$$

Definition 19 (The multilinear extension)**.** *Extremely frequently throughout our construction we refer to objects as both field valued vectors and as field valued multilinear polynomials. In particular, we may refer to a vector $v \in \mathbb{Z}_p^{2^d}$ as a d-variate multilinear polynomial. When interpreting v as a vector, we access the i'th element as v_i. When interpreting v as a multilinear polynomial invoked on input $y \in \mathbb{Z}_p^d$ we write $v(y)$. Therefore interpretation should be clear from notation.*

The natural correspondence between interpretations arises from what we call the ***multilinear extension*** *of the vector v. The multilinear extension is the unique d-variate multilinear polynomial that outputs the i'th element of v on input the binary decomposition of $i-1$. A very important aspect to notice is that vector notation is based on indexing that starts from 1, whereas multilinear evaluation indexing starts from 0, and that is why we needed to first subtract 1 from i above before decomposing it. This 'off-by-one' subtlety is rampant throughout our construction, but unfortunately it is necessary unless we are to change the mathematical convention of indexing vectors starting at 1.*

As far as the explicit formulation of the multilinear extension of a vector v, it arises naturally via the previously introduced Equal *function, see definition 17.*

$$v(\ldots x) \coloneqq \sum_{z \in \{0,1\}^d} \mathrm{Equal}(x, z) \cdot v_{\mathrm{BC}(z)+1}$$

Likewise, given v in multilinear extension form one may infer a vector as follows.

$$\forall i \in 2^d,\ v_i \coloneqq v(\mathrm{BD}(i-1))$$

Using the four definitions above, we not define the variant model of an arithmetic program of the lattice setting.

Definition 20 (Arithmetic program in the lattice setting)**.** *We define an* ***arithmetic program*** *in the lattice setting $(\mathbb{Z}_p, \mathtt{params}, A, B, C)$ as consisting of the five parts in the previous tuple, categorized in three ways. In particular, the first part is the field, the second part is a set of parameters, and the third part is three field valued functions.*

- *The first part $\mathbb{Z}_p$ is a finite field of prime order.*
- *The third part* `params` *is an ordered list of parameters, all natural numbers indicating a size.*

1. $\texttt{constraintCount} \in \mathbb{N}$ *which we abbreviate* $\texttt{Ccount}$.
2. $\texttt{statementSize} \in \mathbb{N}$ *which we abbreviate* $\texttt{Ssize}$.
3. $\texttt{traceSize} \in \mathbb{N}$ *which we abbreviate* $\texttt{Tsize}$.
4. $\texttt{variableSize} \in \mathbb{N}$ *which we abbreviate* $\texttt{Vsize}$.

Often we implicitly access these internal parameters from params *rather than using explicit dot-access notation such as* params.constraintCount.

The last parameter is formulated entirely dependent on the former parameters as follows.

$$\texttt{variableSize} \coloneqq 1 + \texttt{statementSize} + \texttt{traceSize}$$

- *The last three parts are functions referred to as A, B, C expressing the logic of the arithmetic program. The logic functions have the following signature: $A, B, C \colon [\texttt{ConstraintCount}] \times [\texttt{variableSize}] \to \mathbb{Z}_p$. We will interpret these most of the time as multilinear polynomials via their multilinear extensions as described in 19.*

  The three functions A, B, C can be thought of elements of $\mathbb{Z}_p^{\texttt{Ccount} \times \texttt{Vsize}}$, or as matrices over the field $\mathbb{Z}_p$ with ConstraintCount *rows and* variableSize *columns.*

Whereas in the discrete log setting, we separated the definitions of a statement-witness pair, and the satisfaction of an arithmetic program by a statement-witness pair, here we combine the two due to their simpler definitions.

Definition 21 (Arithmetic program satisfaction)**.** *We now define what it means to say a statement $s \in \mathbb{Z}_p^{\texttt{Ssize}}$ together with a trace $t \in \mathbb{Z}_p^{\texttt{Tsize}}$* ***satisfies*** *an arithmetic program in the lattice setting $(\mathbb{Z}_p, \texttt{params}, A, B, C)$.*

First we define the variable vector corresponding to s and t as $(1, s, t) \in \mathbb{Z}_p^{\texttt{Vsize}}$. We also define variable-induced polynomials analogous to those in definition 14 as follows. Crucially, unlike those in definition 14, these are multilinear induced polynomials rather than univariate. They each have the signature $\mathbb{Z}_p^{\texttt{Vsize}} \times \mathbb{Z}_p^{\log(\texttt{Ccount})} \to \mathbb{Z}_p$

$$f_A(v, \ldots x) \coloneqq \sum_{j=1}^{\texttt{Vsize}} A(\ldots x, \mathrm{BD}(j-1)) v_j$$

$$f_B(v, \ldots x) \coloneqq \sum_{j=1}^{\texttt{Vsize}} B(\ldots x, \mathrm{BD}(j-1)) v_j$$

$$f_C(v, \ldots x) \coloneqq \sum_{j=1}^{\texttt{Vsize}} C(\ldots x, \mathrm{BD}(j-1)) v_j$$

where BD *is the binary decomposition function as described in 18.*

Now we say s and t with associated variable vector v satisfy the arithemtic program if the following holds.

$$\forall y \in \{0,1\}^{\texttt{Ccount}},\ f_A(v, \ldots y) \cdot f_B(v, \ldots y) + f_C(v, \ldots y) = 0$$

5.2 Construction

Here we describe our construction of a saga batching scheme in the lattice setting. First we describe the setup of the protocol. Then we describe the starting protocol in subsection 5.2.2 for how the prover and verifier engage regarding proof for a statement s. The starting protocol reduces the task of verification to the task of evaluating a set of polynomials. In order to evaluate the polynomials the prover and verifier enter a recursive protocol for polynomial evaluation which we describe next in subsection 5.2.3. Using the starting protocol together with the protocol for polynomial evaluation reduction, the prover and verifier may engage in any number of proofs, in particular any number polynomial in the security parameter. However, at the end of each proof and before the start of the next, the prover and verifier must engage in an additional protocol described in subsection 5.2.4. The verifier does not make a decision on whether or not to accept any of the proofs, however, until the final protocol described in subsection 5.2.5 is executed.

5.2.1 Setup

The prover and verifier agree on a set of parameters for the protocol as follows.

- The agree on a prime $p \in \mathbb{N}$ defining a finite field $\mathbb{Z}_p$.
- They choose numbers $\ell_1, \ell_2 \in \mathbb{N}$ such that $\ell_1 = 2b\ell_2$ where $b = \lceil \log(\ell_1 + 1) \rceil$. We will assume that ℓ_1 is a power of 2, in particular we define the alias $\eta \coloneqq \log(\ell_1)$.
- They choose a numbers $m, n \in \mathbb{N}$ defining the dimension and the logarithm upper bounding the size of the lattice used, respectively. All parameters with respect to the lattice are discussed in appendix B.
- Uniformly, they agree upon a matrix $R \in_R \mathbb{Z}_p^{m \times n}$ defining a p-ary lattice of dimension m with elements in $\mathbb{Z}_p^n$. As with other vectors, each row of R will also be interpreted as a multilinear polynomial via the natural multilinear extension described in 19.

5.2.2 Starting protocol

We assume that `Ccount`, `Vsize`, and `Tsize`, are powers of two, in particular we define the aliases $\sigma \coloneqq \log(\texttt{Ccount})$, $\tau \coloneqq \log(\texttt{Vsize})$, and $\kappa \coloneqq \log(\texttt{Tsize})$. Then we denote their difference by $d \coloneqq \kappa - \eta$. Note that none of our assumptions regarding parameters as powers of two contradict any existing parameter constraints.

1. The prover holds a statement $s \in \{0,1\}^{\texttt{Ssize}}$ and a corresponding trace $t \in \{0,1\}^{\texttt{Tsize}}$. After committing to the trace the prover will send the statement and the commitment to the verifier. In order to commit to the trace, the prover splits the trace into ℓ_1 binary vectors $W_j \in \{0,1\}^{2^d}$ for $j \in [\ell_1]$. Letting the ℓ_1 vectors W_j form the columns of a binary matrix $W \in \{0,1\}^{2^d \times \ell_1}$, the prover then computes a commitment to the matrix via the matrix product $T = RW$. Note that the data encoded in the commitment T is the entire trace t. The prover sends $s \in \{0,1\}^{\texttt{Ssize}}$ and $T \in \mathbb{Z}_p^{m \times \ell_1}$ to the verifier and proceeds to the next step.

Now we describe exactly how the prover divides t into the vectors W_j for $j \in [\ell_1]$. The vector t is of length $\texttt{Tsize} = 2^\kappa = 2^d \ell_1$. Sequentially from the first bit to the last, every block of 2^d bits form a column of W. That is,

$$\forall j \in [\ell_1],\ W_j = [t_i]_{i \in \{2^d(j-1)+1,\ldots,2^d j\}}$$

This way, upon interpreting the vectors W_j as multilinear polynomials, via their multilinear extensions, see definition 19, we may write t as a multilinear polynomial in terms of them.

$$t(\ldots y) \coloneqq \sum_{z' \in \{0,1\}^\eta} \sum_{z \in \{0,1\}^d} \text{Equal}_\kappa(\ldots y, \ldots z', \ldots z) \cdot W_{(\text{BC}(z')+1)}(\ldots z)$$

We will use this representation in step 8, the final step.

2. The verifier holds on to the commitment matrix T and the statement s and keeps in mind the virtual variable vector $v = (1, s, t)$ which is virtual because the verifier does not know t. We remind the reader that as discussed in subsection 19, we often will refer to the variable vector also as a multilinear polynomial, in particular its natural multilinear extension, see definition 17. The verifier must check that the following equalities hold. They hold if and only if the statement is valid.

$$\forall x \in \{0,1\}^\sigma,\ (f_A(v, \ldots x) f_B(v, \ldots x) + f_C(v, \ldots x)) = 0$$

In order to check them all simultaneously we treat each such expression as a coefficient of a multilinear polynomial, and using the Schwartz-Zippel lemma we check that the multilinear polynomial evaluated at a random point equals zero. To do so the verifier chooses a random input to the multilinear polynomial $\beta_1 \in_R \mathbb{Z}_p^\sigma$ and then the prover and verifier engage in the sumcheck protocol regarding the following sum which represents the evaluation of the multilinear polynomial at input β_1.

$$\sum_{x \in \{0,1\}^\sigma} (f_A(v, \ldots x) f_B(v, \ldots x) + f_C(v, \ldots x)) \cdot \text{Monomial}_\sigma(\ldots x, \ldots \beta_1)$$

3. At the end of the sumcheck protocol in step 1 the verifier is left to evaluate the three induced polynomials at some random point $r_1 \in \mathbb{Z}_p^\sigma$ as well as $\text{Monomial}_\sigma(\ldots r_1, \ldots \beta_1)$. The verifier can evaluate the latter immediately by the succinct representation of Monomial_σ, see definition 16. Regarding the evaluation of the three induced polynomials, the prover sends to the verifier the three claims $e_A, e_B, e_C \in \mathbb{Z}_p$ asserting that

$$e_A = f_A(v, \ldots r_1),\ e_B = f_B(v, \ldots r_1),\ e_C = f_C(v, \ldots r_1)$$

The verifier is left to verifier these evaluation claims.

4. Beyond verifying the three evaluations above, however, there is another item of concern for the verifier. In order for the commitment to be binding, the data in the commitment should be in bits rather than larger numbers. In order to verify this the verifier must check that every variable v_i for $i \in [\texttt{Vsize}]$ satisfies the equation $v_i^2 = v_i$ justifying the

variable to be either 0 or 1. To check that $v_i - v_i^2 = v_i(1-v_i) = 0$ holds for all variables we again treat each such expression as the coefficient of a multilinear polynomial and use the sumcheck to evaluate the multilinear polynomial at a random point. If the evaluation is zero then the verifier can safely conclude that all coefficients are zero and thus all variables are bits. Analogous to the first step, but now with respect to a different domain of summation, the verifier chooses a random scalar $\beta_2 \in_R \mathbb{Z}_p^\tau$ and is left to verify the following sum.

$$\sum_{y \in \{0,1\}^\tau} v(...y)(1 - v(...y)) \cdot \text{Monomial}_\tau(...y, ...\beta_2)$$

Instead of immediately executing the sumcheck regarding this sum, however, we wait until after the next step such that we may bundle this sumcheck together with the sumcheck from the next step. Both sums are over the same domain so they indeed may be bundled into one sum.

5. We now return to verifying the three pending evaluation claims e_A, e_B, e_C with respect to the three induced polynomials on input $r_1 \in \mathbb{Z}_p^\sigma$ left from step 2. To do so the verifier will expand their definitions and bundle them together with a random scalar. The verifier chooses the random scalar $\alpha \in_R \mathbb{Z}_p$ and reduces verifying the three evaluation claims to verifying the following equation holds.

$$\begin{aligned} &e_A + \alpha e_B + \alpha^2 e_C = f_A(v, ...r_1) + \alpha \cdot f_B(v, ...r_1) + \alpha^2 \cdot f_C(v, ...r_1) \\ &= \sum_{y \in \{0,1\}^\tau} \big(A(...r_1, ...y) + \alpha \cdot B(...r_1, ...y) + \alpha^2 \cdot C(...r_1, ...y)\big) v(...y) \end{aligned}$$

6. The previous two steps yielded two sums the verifier must check. We now bundle these two sums into one and use the sumcheck protocol to check it. The verifier chooses a random scalar $\gamma \in_R \mathbb{Z}_p$ and bundles the two sums together as follows and then engages with the prover in a sumcheck protocol over this sum.

$$\begin{aligned} (e_A + \alpha \cdot e_B + &\alpha^2 \cdot e_C) + \gamma \cdot 0 = \\ &\sum_{y \in \{0,1\}^\tau} \big(A(...r_1, ...y) + \alpha \cdot B(...r_1, ...y) + \alpha^2 \cdot C(...r_1, ...y)\big) v(...y) \\ &+ \gamma \sum_{y \in \{0,1\}^\tau} v(...y)(1 - v(...y)) \cdot \text{Monomial}_\tau(...y, ...\beta_2) \end{aligned}$$

Note that since the two sums are over the same summation domain they may be combined into one sum.

7. The verifier must now evaluate the multilinear polynomial v at some random point $r_2 \in \mathbb{Z}_p^\tau$, as well as $A(...r_1, ...r_2), B(...r_1, ...r_2), C(...r_1, ...r_2)$ and $\text{Monomial}_\tau(...r_2, ...\beta_2)$. The latter can be evaluated immediately by the succinct representation of Monomial_τ, see definition 16. Evaluating multilinear polynomials A, B, C will not be done immediately but amortized with their evaluations at other times arising from other proofs

as described in subsection 5.2.4. Suppose the amortization protocol succeeds, leaving the verifier to accept the proof if and only if $v(...r_2)$ evaluates as expected.

Recall that as a vector the variables $v \in \mathbb{Z}_p^{\tau}$ are composed of the constant 1, the statement vector $s \in \mathbb{Z}_p^{\texttt{Ssize}}$, and the trace vector $t \in \mathbb{Z}_p^{\texttt{Tsize}}$ as $v = (1, s, t)$. Using this correspondence we will reduce the evaluation of the multilinear polynomial v at r_2 to evaluations of s and t as multilinear polynomials at r_2.

$$\begin{aligned} v(...y) &= \sum_{z \in \{0,1\}^{\tau}} \text{Equal}_{\tau}(...y, ...z) \cdot v_{(\text{BC}(z)+1)} \\ &= \sum_{n \in \{0,...,\texttt{Vsize}-1\}} \text{Equal}_{\tau}(...y, ...\text{BD}(n)) \cdot v_{n+1} \\ &= \text{Equal}_{\tau}(...y, ...[0]_{i \in \tau}) \cdot 1 \\ &\quad + \sum_{n \in \{1,...,\texttt{Ssize}\}} \text{Equal}_{\tau}(...y, ...\text{BD}(n)) \cdot s_n \\ &\quad + \sum_{n \in \{\texttt{Ssize}+1,...,\texttt{Vsize}\}} \text{Equal}_{\tau}(...y, ...\text{BD}(n)) \cdot t_{(n-\texttt{Ssize})} \end{aligned}$$

Examining this expression we see that in order to evaluate $v(...r_2)$ the verifier may immediately evaluate the first two components as follows due to the fact that the verifier holds the statement s and due to the succinct representation of Equal_{τ}, see definition 17.

$$\text{Equal}_{\tau}(...r_2, ...[0]_{i \in \tau}), \quad \text{and} \sum_{n \in \{1,...,\texttt{Ssize}\}} \text{Equal}_{\tau}(...r_2, ...\text{BD}(n)) \cdot s_n$$

Regarding the last component corresponding to t we will handle that in the next and final step.

8. Recall from step 1 that we may write the multilinear form of t as follows.

$$t(...y) \coloneqq \sum_{z' \in \{0,1\}^{\eta}} \sum_{z \in \{0,1\}^{d}} \text{Equal}_{\kappa}(...y, ...z', ...z) \cdot W_{(\text{BC}(z')+1)}(...z)$$

The verifier is tasked with evaluating t at input r_2. The prover aids the verifier by sending a vector of evaluations $e \in \mathbb{Z}_p^{\ell_1}$ representing the evaluations of W_j for each $j \in [\ell_1]$. That is, the prover asserts that

$$\forall j \in [\ell_1],\ e_j = W_j((r_2)_{\eta+1}, \ldots, (r_2)_{\kappa})$$

The verifier must reduce the task of verifying $t(r_2)$ to the task of verifying the evaluations in e are correct. First note the following equivalence regarding the inner sum

corresponding to $z' = \mathrm{BD}(j-1)$ for $j \in [\ell_1]$ in the representation of the polynomial t.

$$\begin{aligned}
&\sum_{z\in\{0,1\}^d} \mathrm{Equal}_\kappa(...y, ...z', ...z) \cdot W_{(\mathrm{BC}(z')+1)}(...z) \\
&= \sum_{z\in\{0,1\}^d} \mathrm{Equal}_\kappa(...y, ...\mathrm{BD}(j-1), ...z) \cdot W_j(...z) \\
&= \sum_{z\in\{0,1\}^d} \mathrm{Equal}_\eta(y_1, \ldots, y_\eta, ...\mathrm{BD}(j-1)) \cdot \mathrm{Equal}_d(y_{\eta+1}, \ldots, y_\kappa, ...z) \cdot W_j(...z) \\
&= \mathrm{Equal}_\eta(y_1, \ldots, y_\eta, ...\mathrm{BD}(j-1)) \sum_{z\in\{0,1\}^d} \mathrm{Equal}_d(y_{\eta+1}, \ldots, y_\kappa, ...z) \cdot W_j(...z) \\
&= \mathrm{Equal}_\eta(y_1, \ldots, y_\eta, ...\mathrm{BD}(j-1)) \cdot W_j(y_{\eta+1}, \ldots, y_\kappa)
\end{aligned}$$

This implies we may write t as

$$t(...y) = \sum_{j\in[\ell_1]} \mathrm{Equal}_\eta(y_1, \ldots, y_\eta, ...\mathrm{BD}(j-1)) \cdot W_j(y_{\eta+1}, \ldots, y_\kappa)$$

With this equivalence in mind the verifier checks that the evaluations e are as expected by verifying that the following sum equals the expected value of $t(r_2)$ using the succinct representation of Equal_η.

$$\sum_{j\in[\ell_1]} \mathrm{Equal}_\eta((r_2)_1, \ldots, (r_2)_\eta, ...\mathrm{BD}(j-1)) \cdot e_j$$

Thus the verifier has reduced to verifying that the following evaluations are correct.

$$\forall j \in [\ell_1],\ e_j = W_j((r_2)_{\eta+1}, \ldots, (r_2)_\kappa)$$

At this point the prover and verifier enter the recursive protocol for polynomial evaluation reduction, described in the next subsection. As described there, they must enter step 1 with both possessing a tuple of common data $\mathbb{cd}^{(1)}$ and the prover possessing additional prover data $\mathbb{pd}^{(1)}$. Letting $r = [(r_2)_i]_{i\in\{\eta+1,\ldots,\kappa\}}$, and interpreting each column of W as a multilinear polynomial, the data is defined as

$$\begin{aligned}
\mathbb{cd}^{(1)} &= (d \in \mathbb{N}, r \in \mathbb{Z}_p^d, e \in \mathbb{Z}_p^{\ell_1}, T \in \mathbb{Z}_p^{m\times\ell_1}) \\
\mathbb{pd}^{(1)} &= W \in (\mathbb{Z}_p[y_1, \ldots, y_d])^{\ell_1}
\end{aligned}$$

5.2.3 Polynomial evaluation reduction

Following is a three step protocol consisting of a randomization step, followed by a renormalization step, ending in a consistency check between the randomization and the renormalization. There are three states of the protocol consisting of the type of common data held by both the prover and the verifier and the auxiliary data only held by the prover. The randomization step takes the protocol from state 1 to state 2. The renormalization step

takes the protocol from state 2 to state 3. The consistency check step takes the protocol from state 3 back to state 1.

The protocol will be repeatedly executed taking the protocol through states 1,2,3 and back to 1. We use the parameter $d \in \mathbb{N}$ to keep track of the protocol looping. The parameter d is inversely related to the number of times the protocol has been repeated. We actually do not need to count the number of times the protocol is repeated, but rather we can continue looping the protocol until it becomes practical for the verifier to check the pending conditions itself rather than delegate proving the pending conditions to yet another execution of the protocol. The verifier may practically verify the pending conditions on its own when the size of the commitment basis is small enough. Representing the size of the commitment basis is precisely the purpose of d, as it equals the binary logarithm of the size of the commitment basis. Upon repeating the following protocol sufficiently many times until the verifier decides d is small enough, the prover and verifier exit the protocol looping and jump to the protocol described in the next subsection 5.2.4.

In step i of the protocol below we represent the common data and prover data via notation $\mathbb{cd}^{(i)}$ and $\mathbb{pd}^{(i)}$ respectively. We now describe the three states of the protocol.

1. In state 1 the data is formatted as follows.

$$\mathbb{cd}^{(1)} = (d \in \mathbb{N}, r \in \mathbb{Z}_p^d, e \in \mathbb{Z}_p^{\ell_1}, T \in \mathbb{Z}_p^{m \times \ell_1})$$
$$\mathbb{pd}^{(1)} = W \in (\mathbb{Z}_p[y_1, \ldots, y_d])^{\ell_1}$$

   We may interpret each multilinear polynomial in W as a binary vector of length 2^d. The purpose of d is specify this length, namely the length of the binary vectors committed. The commitment is T, each column a commitment to an element of W. The value r represents an input to the polynomials in W, and the vector e represents the corresponding asserted evaluation values. In state 1, the verifier accepts the original statement (no longer needed here), if and only if the commitment T encodes the multilinear polynomials W such that on input r they evaluation to the values e.

2. In state 2 the data is formatted as follows.

$$\mathbb{cd}^{(2)} = (d \in \mathbb{N}, r \in \mathbb{Z}_p^d, e \in \mathbb{Z}_p^{\ell_2}, T \in \mathbb{Z}_p^{m \times \ell_2})$$
$$\mathbb{pd}^{(2)} = W \in \mathbb{Z}_p[y_1, \ldots, y_d]^{\ell_2}$$

   The data may be interpreted as the same as in state 1 except for an important change. Notice how the number of vectors W committed as reduced from ℓ_1 to ℓ_2. The price paid is that now the commitment T is what we call 'degenerate', meaning it is no longer amenable to homomorphic manipulation. The reason is the values committed in the vectors in W have reached a certain magnitude such that if further homomorphic manipulation was performed the commitment would no longer be binding. Analogous to the first state, the verifier accepts the original statement if and only if the multilinear polynomials W encoded in the commitment T evaluate on input r to the evaluations e.

3. In state 3 the data is formatted as follows.

$$\mathbb{cd}^{(3)} = (d \in \mathbb{N}, r \in \mathbb{Z}_p^{d+1}, e \in \mathbb{Z}_p^{\ell_2}, T \in \mathbb{Z}_p^{m \times 2b\ell_2}, T' \in \mathbb{Z}_p^{m \times \ell_2})$$
$$\mathbb{pd}^{(3)} = W \in (\mathbb{Z}_p[y_1, \ldots, y_d])^{\ell_2 \times 2b}$$

We mention how the data differs from the data in state 2. First notice the additional parameter b. This is defined as $b \coloneqq \lceil \log(\ell_1 + 1) \rceil$. It represents the maximum magnitude of values in the degenerate committed data. The commitment T' is actually the same degenerate commitment as in state 2. In contrast, the commitment T is a new *renormalized* commitment that is not degenerate. Both T' and T, however, encode the same committed data. The prover's data W has also been transformed from state 2. Notice it has increased size, in particular from one column to $2b$ columns. Every row, previously carrying only one binary vector (interpreted as such), now carries $2b$ binary vectors, together representing a renormalization of the original degenerate binary vector. In particular, entry corresponding to row j and column k represents the k'th bit of the binary representation of the original j'th degenerate vector values.

We should mention that we do not use the prime notation to indicate degeneracy, but rather to indicate what is new or newly computed in each step of the protocol below. In the case above, T' is a new item in the common data of state 3 not present in state 2.

diagram

Randomization

The randomization step begins with the protocol in state 1 and leaves the protocol in state 2. The common data and prover data are the following.

$$\mathbb{cd}^{(1)} = (d \in \mathbb{N}, r \in \mathbb{Z}_p^{d}, e \in \mathbb{Z}_p^{\ell_1}, T \in \mathbb{Z}_p^{m \times \ell_1})$$
$$\mathbb{pd}^{(1)} = W \in (\mathbb{Z}_p[y_1, \ldots, y_d])^{\ell_1}$$

For the randomization step the verifier sends the prover a challenge and both compute the data for state 2.

1. The verifier chooses a challenge matrix $C \in_R \{0,1\}^{\ell_1 \times \ell_2}$ at uniform. The verifier sends C to the prover.

2. Both the prover and verifier compute $e' \in \mathbb{Z}_p^{\ell_2}$ and $T' \in \mathbb{Z}_p^{m \times \ell_2}$ defined as follows.

$$e'_j = \sum_{k \in [\ell_1]} e_k C_{k,j}$$
$$\forall i \in [m],\ T'_{i,j} = \sum_{k \in [\ell_1]} c_k C_{k,j}$$

The vector e' is the new set of evaluations, and each column of the matrix T' is a degenerate commitment.

3. Now the prover computes the new degenerate data W'. Each column $j \in [\ell_2]$ is computed as follows.

$$W'_j(\ldots y) = \sum_{k \in [\ell_1]} W_k(\ldots y) C_{k,j}$$

In particular, the d-variate polynomials W'_j are multilinear polynomials and on the boolean hypercube of dimension d they only yield values in the set $\{0, \ldots, \ell_1\}$

The prover and verifier exit the randomization step entering state 2 with the following common data and prover data.

$$\mathbb{cd}^{(2)} = (d \in \mathbb{N}, r \in \mathbb{Z}_p^d, e' \in \mathbb{Z}_p^{\ell_2}, T' \in \mathbb{Z}_p^{m \times \ell_2})$$
$$\mathbb{pd}^{(1)} = W' \in \mathbb{Z}_p[y_1, \ldots, y_d]^{\ell_2}$$

Renormalization

The renormalization step begins with the protocol in state 2 and leaves the protocol in state 3. The common data and prover data are the following.

$$\mathbb{cd}^{(2)} = (d \in \mathbb{N}, r \in \mathbb{Z}_p^d, e \in \mathbb{Z}_p^{\ell_2}, T \in \mathbb{Z}_p^{m \times \ell_2})$$
$$\mathbb{pd}^{(2)} = W \in \mathbb{Z}_p[y_1, \ldots, y_d]^{\ell_2}$$

For the renormalization the prover computes data and sends it to the verifier via the following steps.

1. The prover decomposes each vector W_j into b binary vectors. In particular, the prover constructs a matrix $W' \in (\mathbb{Z}_p[y_1, \ldots, y_d])^{\ell_2 \times 2b}$ representing the same data as W but in a renormalized form. In particular, whereas W may be interpreted as an ℓ_2-size vector of $\{0, \ldots, \ell_1\}$-valued vectors, W' may be interpreted as an $\ell_2 \times 2b$-size matrix of $\{0, 1\}$-valued vectors. The prover computes W' according such that is satisfies the following equations.

$$\forall j \in [\ell_2],\ W_j(y_1, \ldots, y_d) = \\ (1 - y_d)\left(\sum_{k \in [b]} W'_{j,2k-1}(y_1, \ldots, y_{d-1}) 2^{k-1} \right) \\ + y_d \left(\sum_{k \in [b]} W'_{j,2k}(y_1, \ldots, y_{d-1}) 2^{k-1} \right)$$

For an explicit formula for computing W' we have the following where $k \in [b]$.

$$W'_{j,2k-1}(y_1, \ldots, y_{d-1}) = \mathrm{BD}(W_j(y_1, \ldots, y_{d-1}, 0))_k$$
$$W'_{j,2k}(y_1, \ldots, y_{d-1}) = \mathrm{BD}(W_j(y_1, \ldots, y_{d-1}, 1))_k$$

Here $\mathrm{BD}(v)_k$ is the k'th least significant bit of the binary decomposition of value $v \in \mathbb{Z}_p$, see definition 18.

2. From W' the prover computes new commitments. Specifically, the prover commits to each of the $\ell_2 2b$ binary vectors in W' via the following formula with $k \in [b]$.

$$T'_{i,j+(2k-1)} = \sum_{y \in \{0,1\}^{d-1}} R(...\mathrm{BD}(i), y_1, \ldots, y_{d-1}, 0) W_{j,k}(y)$$
$$T'_{i,j+2k} = \sum_{y \in \{0,1\}^{d-1}} R(...\mathrm{BD}(i), y_1, \ldots, y_{d-1}, 0) W_{j,k}(y)$$

3. The prover sends the new commitment T' to the verifier.

The prover and verifier exit the renormalization step entering state 3 with the following common data and prover data.

$$\mathbb{cd}^{(3)} = (d - 1 \in \mathbb{N}, r \in \mathbb{Z}_p^d, e \in \mathbb{Z}_p^{\ell_2}, T' \in \mathbb{Z}_p^{m \times 2b\ell_2}, T \in \mathbb{Z}_p^{m \times \ell_2})$$
$$\mathbb{pd}^{(3)} = W' \in (\mathbb{Z}_p[y_1, \ldots, y_d])^{\ell_2 \times 2b}$$

Consistency check

The consistency check step begins with the protocol in state 3 and returns the protocol to state 1. The common data and prover data are the following.

$$\mathbb{cd}^{(3)} = (d \in \mathbb{N}, r \in \mathbb{Z}_p^{d+1}, e \in \mathbb{Z}_p^{\ell_2}, T \in \mathbb{Z}_p^{m \times 2b\ell_2}, T' \in \mathbb{Z}_p^{m \times \ell_2})$$
$$\mathbb{pd}^{(3)} = W \in (\mathbb{Z}_p[y_1, \ldots, y_d])^{\ell_2 \times 2b}$$

For each $j \in [\ell_2]$ the verifier must check the following sums. We describe each sum now separately, but the verifier will bundle them all up together into a single sumcheck protocol.

1. First the verifier must check that all newly committed data W is in bits, the appropriate format for a binding commitment. This way the verifier knows the commitment is binding. To do so the verifier indexes into each of the vectors in the new witness W. To ensure an entry x is a bit, the verifier checks that $x^2 = x$ because only the two identity elements 0 and 1 satisfy that equation.

$$\forall k \in [b], \sum_{y \in \{0,1\}^d} W_{j,(2k-1)}(...y)(1 - W_{j,2k-1}(...y)) = 0$$
$$\forall k \in [b], \sum_{y \in \{0,1\}^d} W_{j,2k}(...y)(1 - W_{j,2k}(...y)) = 0$$

2. Second the verifier must checks that the newly committed data is indeed a renormalization of the previous data. To check this the verifier reconstructs the renormalized data into its degenerate form and uses is to open the previous commitment $T' \in \mathbb{Z}_p^{m \times \ell_2}$. This involves multiplying the bits in the normalized data by powers of 2 to arrive at the degenerate data, and then taking the inner product of the degenerate vectors with the commitment rows of R.

First we introduce the function used to regenerate the degenerate data from the renormalized data. Below is the function W_j which represents the degenerate data in original vector $j \in [\ell_2]$. Notice it is written in terms of the renormalized data. Also note that since the sum is only over size $b = \lceil \log(\ell_1 + 1) \rceil$, the verifier can expand the sum itself, so W_j is really an alias we use for notational convenience. We will use this alias in the next step as well.

$$W_j(y_1, \ldots, y_{d+1}) = \\ (1 - y_{d+1}) \Bigg(\sum_{k \in [b]} W_{j,(2k-1)}(y_1, \ldots, y_d) 2^{k-1} \Bigg) \\ + y_{d+1} \Bigg(\sum_{k \in [b]} W_{j,2k}(y_1, \ldots, y_d) 2^{k-1} \Bigg)$$

With the vectors in hand W_j for $j \in [\ell_2]$ which supposedly hold the degenerate data contained in the degenerate commitments T', the verifier now uses the following sums to check that for each degenerate data vector W_j opens to the correct degenerate commitment which is the j'th column of T.

$$\forall x \in \{0,1\}^{\lceil \log(m) \rceil},\ T'_{\mathrm{BC}(x),j} = \\ \sum_{y \in \{0,1\}^{d+1}} R\big(\ldots x, \ldots y, \ldots [0]_{i \in [n-(d+1)]}\big) W_j\big(\ldots y, \ldots [0]_{i \in [n-(d+1)]}\big)$$

3. Third the verifier must check that the pending evaluation conditions on the degenerate data are correct. To do so, the verifier again uses the regenerated degenerate data vectors provided by the aliases W_j and evaluates them on the multilinear point $r \in \mathbb{Z}_p^{d+1}$ That is, for all $j \in \ell_2$ we must have

$$e_j = \sum_{y \in \{0,1\}^{d+1}} W_j(\ldots y) \cdot \mathrm{Monomial}(\ldots y, \ldots r)$$

In order to bundle all of the above sums together into single sum for a single sumcheck protocol, they must be over the same domain. Note that all sums are either over the domain $\{0,1\}^d$ or $\{0,1\}^{d+1}$. We will use the larger domain of the sumcheck, and any sum over the smaller domain may be augmented with an extra variable. In particular, we reformulate the sums for the bit checks as follows.

$$\forall k \in [b], \sum_{z \in \{0,1\}} z \sum_{y \in \{0,1\}^d} W_{j,(2k-1)}(\ldots y)(1 - W_{j,2k-1}(\ldots y)) = 0$$
$$\forall k \in [b], \sum_{z \in \{0,1\}} z \sum_{y \in \{0,1\}^d} W_{j,2k}(\ldots y)(1 - W_{j,2k}(\ldots y)) = 0$$

To bundle all the sums for a single sumcheck the verifier chooses a random scalar $\gamma \in_R \mathbb{Z}_p$ and plugs γ into the univariate polynomial in which each sum corresponds to a coefficient.

Finally the prover and verifier engage in a sumcheck protocol over the bundled sum. At the end, the verifier is left to perform the following evaluations for some random value $r' \in \mathbb{Z}_p^d$

$$\begin{gathered}\text{Monomial}(...r', ...r) \\ \forall x \in \{0,1\}^{\lceil \log(m) \rceil},\ R\big(...x, ...r', ...[0]_{i \in [n-(d+1)]}\big) \\ \forall j \in [\ell_2],\ \forall k \in [b],\ W_{j,(2k-1)}(...r'),\ W_{j,2k}(...r')\end{gathered}$$

Due to the succinct representation of Monomial, the verifier may immediately compute the first evaluation. Just like with the multilinear polynomials A, B, C, the verifier does not immediately evaluate the m evaluations on R but rather delegates them to the process described in 5.2.4. Regarding the last $\ell_2 2b$ evaluations, these are the evaluations embedded in the common data for the next step, and are thus delegated to the next step.

The prover and verifier exit the consistency checking step reentering state 1 with the following common data and prover data.

$$\begin{aligned}\mathbb{cd}^{(1)} &= (d \in \mathbb{N}, r' \in \mathbb{Z}_p^d, e' \in \mathbb{Z}_p^{2b\ell_2}, T \in \mathbb{Z}_p^{m \times 2b\ell_2}) \\ \mathbb{pd}^{(1)} &= W \in (\mathbb{Z}_p[y_1, \ldots, y_d])^{\ell_2 \times 2b}\end{aligned}$$

5.2.4 Ending a proof

When a prover and verifier engage in the starting protocol followed by the protocol for polynomial evaluation reduction, the parameter d grows smaller with every execution of polynomial evaluation reduction. At some point, the verifier decides it is ready to handle the remaining polynomial evaluations manually rather than delegating them to yet another execution. When this time occurs, the prover and verifier engage in the following protocol. Specifically, we require that the prover and verifier enter this protocol only when they are in state 2 of the polynomial evaluation reduction protocol. Either of the other states would work as well, state 1 more than state 3, but state 2 allows for the most efficiency. As such, they enter this protocol with the following common data and prover data.

$$\begin{aligned}\mathbb{cd}^{(2)} &= (d \in \mathbb{N}, r \in \mathbb{Z}_p^d, e \in \mathbb{Z}_p^{\ell_2}, T \in \mathbb{Z}_p^{m \times \ell_2}) \\ \mathbb{pd}^{(2)} &= W \in \mathbb{Z}_p[y_1, \ldots, y_d]^{\ell_2}\end{aligned}$$

The prover and verifier execute the following steps.

1. The prover sends W to the verifier.

2. Interpreting each element of W as a field-valued vector, the verifier checks that each vector is of length at most 2^d. Secondly, the verifier also checks that each value of each vector has magnitude at most ℓ_1. Thirdly, the verifier checks that each vector opens to the corresponding commitment in T. That is, the verifier checks that the following equalities hold.

$$\forall j \in [\ell_2],\ \forall i \in [m],\ T_{i,j} = \sum_{k=1}^{n} R_{i,k} \cdot (W_j)_k$$

3. Interpreting each element of W as a multilinear polynomial, the verifier checks that on input r they evaluate to the claimed evaluations in e. That is, the verifier checks that the following equalities hold.

$$\forall j \in [\ell_2],\ W_j(...r) = e_j$$

If the checks performed by the verifier above all pass, then the verifier continues to the following routine for amortizing the pending evaluations of multilinear polynomials A, B, C, and R. Recall that from step 7 of the starting protocol, the verifier is left to evaluate $A(...r_1, ...r_2), B(...r_1, ...r_2), C(...r_1, ...r_2)$ for random inputs $r_1 \in \mathbb{Z}_p^\sigma$ and $r_2 \in \mathbb{Z}_p^\tau$. Also recall that at the end of each consistency check for the polynomial evaluation reduction protocol, the verifier is left to evaluate the following for some point $r' \in \mathbb{Z}_p^d$.

$$\forall x \in \{0,1\}^{\lceil \log(m) \rceil},\ R\big(...x, ...r', ...[0]_{i \in [n-(d+1)]}\big)$$

Rather than doing so directly, the verifier amortizes all of these evaluations via the **EvaluationReduction** protocol described in 3.4.2. Specifically, after the second proof, there will be two such sets of evaluations the verifier must evaluate. Via the evaluation reduction protocol, each pair of evaluations is amortized, reducing to a single evaluation. The number of pairs is $3 + m$ so the evaluation reduction protocol will be performed $3 + m$ times. In general, at the end of the i'th proof for $i \geq 2$, there will be two sets of evaluations to perform, and they will all be amortized via the evaluation reduction protocol.

5.2.5 Final protocol

At the end, after the prover has submitted all proofs, the verifier has one final set of checks to perform. The verifier must manually verify the final set of $3 + m$ evaluations that have been amortized via the **EvaluationReduction** protocol up to this final point. Suppose the evaluations are $A(...r_1, ...r_2), B(...r_1, ...r_2), C(...r_1, ...r_2)$ for random inputs $r_1 \in \mathbb{Z}_p^\sigma$ and $r_2 \in \mathbb{Z}_p^\tau$, together with the following set of evaluations for some point $r' \in \mathbb{Z}_p^d$.

$$\forall x \in \{0,1\}^{\lceil \log(m) \rceil},\ R\big(...x, ...r', ...[0]_{i \in [n-(d+1)]}\big)$$

The verifier manually performs these evaluations and compares with the expected values. If all evaluations match, the prover accepts all proofs sent by the prover, and otherwise rejects all proof. This completes the lattice based scheme for saga batching.

5.3 Protocol properties

Theorem 7 (Construction 5.2 is a saga batching scheme)**.** *The construction presented in section 5.2 is a valid saga batching scheme. Let the prime p defining the field $\mathbb{Z}_p$ be chosen such that $p = \text{poly}(2^\lambda)$ for security parameter λ. Furthermore, let $2n\sqrt{\ell_1} < p$, and $m = \lambda$.*

Completeness: *If a prover is honest and follows the protocol, only trying to prove true statements with valid witnesses, the verifier accepts all proofs in the batch with probability* 1.

Soundness: *If a prover is dishonest, does not follow the protocol or tries to prove false statements, the verifier rejects all proofs in the batch with probability at least* $\text{poly}(\lambda)/p$.

Proof of knowledge: *When the verifier performs randomization on commitments, reducing the number of commitments the verifier must check, the following proof of knowledge property holds. If the prover knows an opening for every output commitment of the randomization, then the prover knows an opening for every input commitment to the randomization. Furthermore, the prover can extract openings to the input commitments from the openings of the output commitments.*

Space complexity: *Examining the construction, one may observe the following space complexity parameters.*

1. *The* Initial communication size *consists of the lattice, which is of size* $\log(p)^{mn}$.
2. *The* Per proof size *is poly-logarithmic in the witness size, the precise number depending on the number of recursions execution of protocol 5.2.3.*
3. *The* Final communication size *is a constant number of field elements sent by the prover in protocol 5.2.5.*

5.3.1 Completeness and Soundness

Completeness follows naturally from the construction. Soundness holds via the construction in combination with the soundness of the sumcheck protocol described in 3.4.3 and the polynomial evaluation protocol described in 3.4.2. In particular, every randomized reduction performed by the verifier is either in regard to the lattice commitment which is of concern for proof of knowledge rather than soundness, or in regard to the sumcheck protocol of the polynomial evaluation reductions performed at the end of each proof as described in 5.2.4.

5.3.2 Zero-knowledge

We have constructed a saga batching scheme in the lattice setting without zero-knowledge. This is due to the significant additional complexity that zero-knowledge would incur. This is not to say zero-knowledge can't be applied to our construction, and indeed we now describe briefly how it may be done.

First of all, just like in the discrete log setting, at the end the simulator will need to open commitments. As we did with discrete logs, we allow the simulator to do this by providing it with a trapdoor. Such a trapdoor for the SIS problem can be found in literature such as [MP11] and [GM17]. Also, the commitments themselves would need to be masked but this can be easily established by an information theoretic argument or the leftover hash lemma by extending the commitment size n to make room for masks. In particular, if the group defining the lattice is $\mathbb{G}$ then for statistical zero-knowledge we must allow for an addition $\log(\mathbb{G})$ bits.

In each proof, the three induced polynomials are evaluated at a single point. This must be compensated for by randomizing the induced polynomials each with one degree of freedom.

To do so, the variable vector is has three variants just like in the discrete log setting, and each is augmented with a mask that is transfered to the induced polynomials via the likes of $H'_A, H'_B, H'_C \subseteq H$ in the discrete log setting.

The complexity of zero-knowledge in the lattice setting, however, comes from the recursive process of revealing evaluations of the hidden witnesses and then normalizing them, and then performing consistency checks further revealing evaluations. Every renormalization would need to include further randomization to account for these evaluations, and as such we could not simply cut the basis length in half in each successive protocol but each would more than half the length of the previous to make room for masks. Perhaps most inconvenient of all, the indexing notation would become far more complex. Already complex enough, the indices of summation would need to jump over the masks in the many of the summations because they are not to be evaluated over, but not in all because they must still be checked to be in bit format.

The randomization discussed so far is only with respect to the commitments and the evaluations, but we must also make the sumcheck protocol zero knowledge. This may be done efficiently following the construction in [XZZ+19], in which the prover commits to a low degree randomized polynomial with exactly as many coefficients as degrees of freedom revealed in the sumcheck protocol. Then the verifier issues a random scalar and the sumcheck is subsequently performed over the random combination of the original sum and the sum over the randomized commitment. We mention that other works such as [Set19] and its precedent [WTS+18] alternatively make the sumcheck protocol zero-knowledge by performing it homomorphically over committed data using discrete logs. This alternative is significantly more expensive in computing time. A variant in the lattice setting is not known to exist, though if it did it would be significantly more expensive than the simpler option presented in [XZZ+19].

5.3.3 Proof of knowledge

Our core proof of knowledge for lattices comes from [BBC+18], one of the few previous works using lattices to construct practical proof systems in the sense that the verifier does less work verifying than it would without help from the prover.

By the selection of parameters in theorem 5, the commitments remain binding even after randomization. In particular, the norm of the commitment is the Euclidean norm, and by means of the construction will be at most $\beta = n\sqrt{\ell_1} < p/2$. However, the proof can only exctract a witness of 2β, thus lattice parameters must be increased to account for this slack. Together with the dimension of the lattice, $m = \lambda$, this implies security. See appendix B discussing the selection of parameters for the lattice commitment scheme.

First we generalize a counting argument from [Dam10].

Lemma 7.1 (Heavy Table lemma)**.** *Consider a table $\mathcal{T}$ of truth values, with n rows and m columns. Suppose $\mathcal{T}$ has an ϵ fraction of true entries, that is ϵnm true entries. For $t \in [0, 1]$, define the **heavy** rows of $\mathcal{T}$ to be those containing at least an ϵt fraction of true entries, that is at least ϵtm true entries. Then more than a $1 - t$ fraction of all true entries, that is more than $(1 - t)nm$ true entries, exist in heavy rows.*

Proof. Let α be the number of true entries in heavy rows and β the number of true entries in non-heavy rows, noting $\alpha + \beta = \epsilon nm$. Every non-heavy row contains less than ϵtm true entries. Summing over all $n' \leq n$ non-heavy rows we have $\beta < n'(\epsilon tm) \leq n(\epsilon tm)$. Then we have a lower bound on alpha as

$$\alpha = \epsilon nm - \beta > \epsilon nm - n(\epsilon tm) = \epsilon nm(1-t)$$

Thus the fraction of true entries that exist in heavy rows is $\alpha/(\epsilon nm) > 1 - t$. □

In order to prove our proof of knowledge for lattices we simplify notation, and rather than doing the analysis with respect to the typical lattice problem called the 'short integer solution' problem (SIS for short) in which the group is a $\mathbb{Z}_q^m$, we abstract the problem into one for abelian groups in general. A similar abstraction was proposed previously in [GINX14].

Definition 22 (GSIS Problem)**.** *The* Group Short Integer Solution (GSIS) *problem, denoted by GSIS*$(\mathbb{G}, g, \beta, p)$*, is the following. Given finite abelian group* $\mathbb{G}$ *and uniformly selected* $g \in \mathbb{G}^n$*, find* $x \in \mathbb{Z}^n$ *such that* $x \neq 0$*,* $\|x\|_p \leq \beta$*, and* $\prod_{i=1}^n g_i^{x_i} = id_{\mathbb{G}}$ *where* $id_{\mathbb{G}}$ *is the identity element of* $\mathbb{G}$*.*

We will use the Euclidean norm, that is p-norm 2. Let us clarify the objects of analysis in this proof.

- $g \in \mathbb{G}^n$ is the vector of group constants used for the commitment.
- $S \in \mathbb{Z}^{n \times \ell_1}$ is the matrix of small coefficients that represent the data to which the prover commits.
- $C \in \{0,1\}^{\ell_1 \times \ell_2}$ is the challenge issued by the verifier to check the integrity of the prover's commitment.
- $T \in \mathbb{G}_1^{\ell}$ is the vector of group elements representing the commitment made by the verifier.

To avoid the subleties of the algebra of linear maps between spaces of different types, that is matrices with elements of different types, we avoid the shorthand convenience of matrix notation and instead explicitly use summations. We use two different types of elements, one type being integers perhaps modulo some number, the other type being group elements of some abelian group. To avoid the difficulty of reading complex exponents, we avoid multipliative notation and instead exmploy additive notation which is appropriate because the arbitrary group is assumed abelian. In order to distinguish between integer elements and group elements, we write the integers on the left, followed, by a center dot, followed by the group element on the right. For example, for group element g and integer ι we write $\iota \cdot g$ to repesent the ι-multiple of the group element g.

Theorem 8 (Proof of knowedge for lattice commitments)**.** *Suppose the prover succeeds in passing the verifier's challenge with probability* $\epsilon \geq 1/2^{\lambda}$*. Then in expected time* $\text{poly}(\lambda)/\epsilon$ *the prover can extract a witness for the original commitment of twice the norm, that is* $2\ell_1\beta$*.*

In particular, the probability of the prover's success in passing the verifier's challenge is taken of not just the random sampling of the verifier's challenge, but also over the internal coin tosses used by the prover in its effort to find a witness for the challenge.

Proof. We construct an extractor $\mathcal{E}_{j'}$ for each $j' \in \{1, \ldots, \ell_1\}$ in order to extract a witness corresponding to group element j' of the ℓ_1 group elements in the original commitment, that is a witness of the form $s \in \mathbb{Z}^n$ such that $\sum_{i=1}^{n} s_i \cdot g_i = t_{j'}$. Then we combine the ℓ_1 extractors into a single extractor that achieves the promised properties.

We first describe the purpose of extractor $\mathcal{E}_{j'}$ in detail. We claim that $\mathcal{E}'_j$ runs in expected time $\text{poly}(\lambda)/\epsilon$ and outputs a vector $s \in \mathbb{Z}^n$ such that the following two conditions hold.

- The first condition captures the fact that the extracted witness s indeed corresponds to the appropriate target $t_{j'}$ in that it satisfies the following equation.

$$\sum_{i=1}^{n} s_i \cdot g_i = t_{j'}$$

- The second condition captures the fact that the extracted witness s is valid in the sense that its norm remains small to the extent that it satisfies the following equation.

$$\big(\sum_{i=1} s_i^2\big)^{1/2} \leq 2\ell_1\beta$$

Given all $j' \in \{1, \ldots, \ell_1\}$ extractors $\mathcal{E}_{j'}$ together they constitute a single extractor $\mathcal{E}$ that achieves expected running time $\text{poly}(\lambda)/\epsilon$ and outputs a witness S To conclude the proof, we show that we may combine the ℓ_1 extractors given above into a single extractor such that the extractor runs in expected time $\text{poly}(\lambda)/\epsilon$ and extracts a master witness $S \in \mathbb{Z}^{m \times \ell_1}$ with column j' as the witness extracted by extractor $\mathcal{E}_{j'}$. Thus proof of knowledge holds as described in the theorem statement.

Construction: Here we describe the construction of the extractor $\mathcal{E}_{j'}$. Let the program of $\mathcal{E}_{j'}$ consist of repeating the following subroutine $\mathcal{O}(\lambda)$ times or until a valid witness is found. The hidden constant in $\mathcal{O}(\lambda)$ reflects the cost of a brute force search for a witness, which depends on n and the order of the group, but for concreteness the constant $n/2$ is conservatively sufficient as described later. If any trial of the subroutine outputs a witness then let $\mathcal{E}_{j'}$ output that witness. If all λ trials result in the failure symbol $\perp$ then let $\mathcal{E}_{j'}$ also output $\perp$.

1. Sample a random challenge $C^{(1)} \in_R \{0,1\}^{\ell_1 \times \ell_2}$ and a random auxiliary input for the adversary $r \in_R \{0,1\}^r(\lambda)$.

   Feed the sampled challenge and random auxiliary input to the adversary and let the adversary's program begin to seek a witness for the challenge using the auxiliary input as its source of random coins. We may express the invocation as $\mathcal{A}(C^{(1)}, r)$.

   If the adversary's program succeeds and outputs a valid witness $S^{(1)} \in \mathbb{Z}^{n \times \ell_2}$, continue to the next step. If the adversary's program fails and outputs an invalid witness or no witness at all, repeat this step.

2. Sample another random challenge $C^{(2)} \in_R \{0,1\}^{\ell_1 \times \ell_2}$ with the constraint that row j remain the same as in challenge $C^{(1)}$. That is, $\forall k \in \{1, \ldots, \ell_2\}, C^{(1)}_{j',k} = C^{(2)}_{j',k}$. In other words, this is equivalent to sampling a random challenge in the set $\{0,1\}^{\ell_1 - 1 \times \ell_2}$.

Feed the newly sampled challenge and the original random auxiliary input to the adversary and let the adversary's program again seek a witness. We may express the invocation as $\mathcal{A}(C^{(2)}, r)$.

Repeat this step at most λ/ϵ times or until the the adversary returns a success. If the adversary's program eventually succeeds and outputs a valid witness $S^{(2)} \in \mathbb{Z}^{m \times \ell_2}$, continue to the next step. If the adversary's program fails for all λ/ϵ tries, abort the with the failure symbol $\perp$.

3. At this point the extractor has extracted two witnesses $S^{(1)}, S^{(2)} \in \mathbb{Z}^{n \times \ell_2}$ corresponding to two challenges $C^{(1)}, C^{(2)} \in_R \{0,1\}^{\ell_1 \times \ell_2}$ such that the challenges differ only in row j'. From this the extractor computes the desired witness $s \in \mathbb{Z}^n$ as

$$\forall i \in \{1, \ldots, n\}, s_i = S^{(1)}_{i,k'} - S^{(2)}_{i,k'}$$

Analysis: Here we prove the construction of $\mathcal{E}_{j'}$ described directly above indeed satisfies its promised properties, that its promised expected running time and its promised witness output. First we prove the validity of the witness output by $\mathcal{E}_{j'}$. Second we prove that $\mathcal{E}_{j'}$ runs in expected time $\text{poly}(\lambda)/\epsilon$. Third we prove that $\mathcal{E}_{j'}$ will always output a witness.

Proof of validity: Now regarding the validity of extracted witness s. Since rows j' of $C^{(1)}$ and $C^{(2)}$ are different, there must exist a k' for which $C^{(1)}_{j',k'} \neq C^{(2)}_{j',k'}$ and for all $k \neq k'$ we have $C^{(1)}_{j',k} = C^{(2)}_{j',k}$. Without loss of generality we may say $C^{(1)}_{j',k'} - C^{(2)}_{j',k'} = 1$ because should it equal -1 we may simply use negation. In particular, we may substitute $-s$ for s.

$$\sum_{i=1}^{n} S^{(1)}_{i,k'} \cdot g_i = \sum_{k=1}^{\ell_2} C^{(1)}_{j',k} \cdot t_{j'}$$
$$\sum_{i=1}^{n} S^{(2)}_{i,k'} \cdot g_i = \sum_{k=1}^{\ell_2} C^{(2)}_{j',k} \cdot t_{j'}$$
$$\implies$$
$$\sum_{i=1}^{n} (S^{(1)}_{i,k'} - S^{(2)}_{i,k'}) \cdot g_i = \sum_{k=1}^{\ell_2} (C^{(1)}_{j',k} - C^{(2)}_{j',k}) \cdot t_{j'} = (C^{(1)}_{j',k'} - C^{(2)}_{j',k'}) \cdot t_{j'} = 1 \cdot t_{j'} \qquad \implies$$
$$\sum_{i=1}^{n} s_i \cdot g_i = t_{j'}$$

The above demonstrates that s satisfies the first condition promised by extractor $\mathcal{E}_{j'}$.

We must also show that s satisfies the second condition promised by the extractor, that is the condition regarding the bounded norm of s. Recall that by assumption every column of $S^{(1)}$ and $S^{(2)}$ has Euclidean norm bounded by $\ell_1 \beta$. We may then employ the Minkowski inequality to bound the Euclidean norm of their difference. Specifically, we write the following wherein the equality follows from the Minkowski inequality and $S^{(1)}_{\cdot,k'}$ denotes

the k'th column of $S^{(1)}$ and likewise for $S^{(2)}$.

$$||s||_2 = \big(\sum_{i=1}^{n} s_i^2\big)^{1/2} = \big(\sum_{i=1}^{n}(S^{(1)}_{i,k'} - S^{(2)}_{i,k'})^2\big)^{1/2}$$
$$\leq \big(\sum_{i=1}^{n}(S^{(1)}_{i,k'})^2\big)^{1/2} + \big(\sum_{i=1}^{n}(S^{(2)}_{i,k'})^2\big)^{1/2} = ||S^{(1)}_{\cdot,k'}||_2 + ||S^{(1)}_{\cdot,k'}||_2 = 2\ell_1\beta$$

Proof of running time: Regarding the running time of the extractor, we first examine the running time of the subroutine. The first step of the subroutine must be repeated an $1/\epsilon$ expected number of times before reaching the second step, because by assumption the adversary succeeds with probability ϵ over the sample space sampled in this first step. The second step is executed at most λ/ϵ times, and the third step requires execution at most $\mathcal{O}(\lambda)$ times. Therefore the expected running time of the subroutine is $\mathcal{O}(\lambda)/\epsilon$. Since there are at most $\mathcal{O}(\lambda)$ trials of the subroutine the total expected running time is $\text{poly}(\lambda)/\epsilon$ as claimed in the theorem statement.

Proof of output: Regarding the guaranteed output of the extractor $\mathcal{E}_{j'}$, we prove below the failure probability of the subroutine is less than $0.75 + 0.61^\lambda$. Therefore upon executing the subroutine $\mathcal{O}(\lambda)$ times the final failure probability of $\mathcal{E}_{j'}$ is bounded from above by $(0.75 + 0.61^\lambda)^{\gamma\lambda}$ for some constant γ. One must choose γ such that the failure probability is less than or equal to the probability of success on random sampling, that is the probability of success on a brute force search. In other words, the number of possible witnesses, which is conservatively bounded from above by 2^m, is at most the reciprocal of the probability of failure, which in turn is less than number of total samples attempted by the extractor. If the extractor samples more times then their are possible witnesses, the extractor is effectively performing a brute force search. Since a brute force search never fails, we may conclude that $\mathcal{E}_{j'}$ never fails to provide a witness. If one wishes to compute an explicit bound on γ, one may do so as follows.

$$(0.75 + 0.61^\lambda)^{\gamma\lambda} \leq 1/2^m$$
$$\implies \gamma\lambda \log_2(0.75 + 0.61^\lambda) \leq -m$$
$$\implies \gamma \leq \frac{-m}{\lambda \log_2(0.75 + 0.61^\lambda)}$$
$$\implies \gamma \leq m/20 \text{ for } \lambda \geq 50$$

where the last inequality can be verified numerically.

Now we must prove the subroutine fails with probability at most $0.75 + 0.61^\lambda$. The failure of the subroutine is dependent on the failure of the adversary's program to find witnesses when being invoked with random challenges and random auxiliary inputs by the subroutine. When we discuss the probability of the adversary succeeding we are talking about a probability space in which the sample space consists of the set $(C, r) \in \{0,1\}^{\ell_1 \times \ell_2} \times \{0,1\}^{r(\lambda)}$, and the outcome space consists of the truth values of whether or not the adversary succeeded on the given sample. The adversary is a random variable mapping the sample space to the truth values. In order to analyze this random variable, we model it as a truth

table, where each entry corresponds to an element of the sample space, and the value of the entry indicates whether the adversary is successful on the sample corresponding to the entry. Specifically, consider a truth table $\mathcal{T}$ in which we make the following identifications.

- We identify the columns of $\mathcal{T}$ with the 2^{ℓ_2} possibilities for row j' of C.
- We identify the rows of $\mathcal{T}$ with the remaining degrees of freedom of the sample space. In particular, there are $\ell_1 - 1$ remaining rows of C each of which has 2^{ℓ_2} possibilities, and there is the random auxiliary input $r \in \{0,1\}^{r(\lambda)}$ which has $2^{r(\lambda)}$ possibilities. Thus there are $2^{\ell_2(\ell_1-1)}2^{r(\lambda)}$ remaining possibilities, each of which we identify with a unique row of $\mathcal{T}$.
- We identify entry (i, j) of $\mathcal{T}$ with the truth value indicating whether or not the adversary succeeds on the challenge and random auxiliary input identified with row i and column j.

Suppose we define a row of $\mathcal{T}$ to be *heavy* if it contains an ϵt fraction of true entries for some $t \in [0, 1]$. Then the Heavy Table theorem assert that more than an $1 - t$ fraction of all true entries are located in heavy rows. We would like the heavy rows to be those in which we might find two true entries, that way finding two witnesses corresponding to two challenges only differing in row j'. Suppose upon sampling a first challenge $C^{(1)}$ and auxiliary input r the adversary returns a valid witness. Since more than a $1 - t$ fraction of all true entries are located in heavy rows, our sampling is located in a heavy row with probability more than $1-t$. Now that we are in a heavy row, at least an ϵt fraction of entries in the row are true, that is at least $\epsilon t 2^{\ell_2}$ entries in the row are true. Our current sampling (C, r), however, is already one of these true values, so there are $\epsilon t 2^{\ell_2} - 1$ remaining true values in the selected row. Therefore upon randomly sampling other entries in the row by sampling second challenges $C^{(2)}$, we will land upon a second true entry with probability at least $(\epsilon t 2^{\ell_2} - 1)/2^{\ell_2}$.

We establish four probabilities in the list below and then we use these four probability in a conditional probability equation to express the probability the subroutine fails.

- Since more than a $1 - t$ fraction of all true entries are located in heavy rows, the probability the selected row is heavy is more than $1 - t$. Therefore we have

$$\Pr(\text{Heavy}) \leq 1 \text{ and } \Pr(\text{Not Heavy}) \leq 1 - (1 - t) = t$$

- If the selected row is heavy then a random second sampling will be successful with probability at least

$$\begin{aligned}(\epsilon t 2^{\ell_2} - 1)/2^{\ell_2} &= \epsilon t - 1/2^{\ell_2} \\ &\geq \epsilon t - 1/2^{\lambda+2} = \epsilon t - (1/2^{\lambda})(1/2^2) \\ &> \epsilon t - \epsilon/4 = \epsilon(t - 1/4)\end{aligned}$$

  where the first inequality follows from the assumption $\ell_2 \geq \lambda + 2$ and the second inequality follows from the assumption that $\epsilon > 1/2^{\lambda}$. Therefore upon repeating step two λ/ϵ times the probability of failure when the selected row is heavy is bounded from above by $(1 - \epsilon(t - 1/4))^{\lambda/\epsilon}$.

- The subroutine will certainly fail if the selected row is not heavy because then the row only contains the current success and no other, therefore $\Pr(\text{Fail}|\text{Not Heavy}) = 1$.

Putting these four probabilities together we bound the probability of the subroutine failing $\Pr(\text{Fail})$ as follows.

$$\begin{aligned}\Pr(\text{Fail}) &= \Pr(\text{Fail}|\text{Heavy})\Pr(\text{Heavy}) + \Pr(\text{Fail}|\text{Not Heavy})\Pr(\text{Not Heavy}) \\ &< (1-\epsilon(t-1/4))^{\lambda/\epsilon} * 1 + 1 * t\end{aligned}$$

Only now, as late as possible, do we select a value for $t \in [0,1]$ such that it is clear we did not lose any opportunity by a premature selection t.

Remark 3. *We comment that the proof from [BBC+18] uses the value $t = 1/2$ because it uses the heavy table lemma from [Dam10] which is specialized to $t = 1/2$, and as a result some equations in the proof are flawed, yet the idea of the proof holds. In contrast we prove the heavy table lemma for all $t \in [0,1]$ such that we may select an appropriate value when needed.*

Strictly speaking we should choose the value of $t \in [0,1]$ with the goal of minimizing the upper bound, because the upper bound we choose determines the constant γ, that is the number of times we execute the subroutine. But γ is a constant so its concrete value does not matter for asymptotic analysis. Therefore we relax our choice of t to be roughly appropriate. Suppose we choose $t = 3/4$. Upon plugging in $t = 3/4$ we see the expression $(1-\epsilon/2)^{\lambda/\epsilon}$ assumes its maximum as $\epsilon \to 0$ and can numerically be seen to be bounded above by 0.61^{λ}. Thus we are left with the following upper bound as we intended to prove.

$$\Pr(\text{Fail}) < 0.61^{\lambda} + 0.75$$

This completes the proof of knowledge. □

Chapter 6

Conclusion

In this work we have reviewed the history and contributed to the field of interactive proof systems. First we compared the two founding papers of interactive proof systems, [GMR85] and [Bab85]. Then we addressed one of the first major questions facing the field, that is the difference between public coin and private coin proofs. We outlined how the work [GS86] answered this question by showing how the two are equivalent. Next we explored the alternative model of interactive proofs involving multiple provers. Analogous to the famous result that every language computable in polynomial space has an interactive proof system, every language computable in nondeterministic exponential time has an multi-prover interactive proof system. Lastly we reviewed the definitions underlying the concept of zero-knowledge, as well as alternative definitions that have been proposed.

Using the ideas we reviewed, in particular the concepts of interactive proofs and zero-knowledge, we provide two novel constructions of our own. As motivation, we propose a new scenario in which the many proofs are to be batched together at amortized cost with minimal space complexity for both the prover and verifier. Such a scenario may occur, for example, in the context of a smart contract that operates on a blockchain and must verify many proofs without consuming much space due to high storage costs of permanent data on blockchains. We call a solution to such a scenario a saga batching scheme because its as if the prover tells the verifier a long story over time and at the end the verifier decides whether or not to accept all the proofs sent by the prover, as if its the last punchline of the saga.

We construct two saga batching schemes, one using cryptography from discrete logs and one using cryptography from lattices. Both are efficient in allowing an arbitrary number of proofs to be proven with minimal overhead for the verifier until the end when the verifier makes a final computation to determine its decision. The discrete log construction becomes practical with small size proofs but is vulnerable to quantum cryptanalysis, whereas the lattice construction only becomes practical with large size proofs but is resistant to quantum cryptanalysis. Both constructions provide insights that may be of independent interest. The discrete log construction uses a new form of amortization that prevents the need to commit to additional redundant data as done in previous constructions. The lattice construction uses a recursive process of randomizing then normalizing the data to overcome the limiting overflow problems present in previous lattice constructions.

Bibliography

[Ajt96] M. Ajtai. Generating hard instances of lattice problems. *Electron. Colloquium Comput. Complex.*, 3, 1996.

[Bab85] L. Babai. Trading group theory for randomness. In *STOC '85*, 1985.

[BBB+18] Benedikt Bünz, Jonathan Bootle, D. Boneh, A. Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.

[BBC+18] C. Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. *IACR Cryptol. ePrint Arch.*, 2018:560, 2018.

[BCC87] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37:156–189, 1987.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. *IACR Cryptol. ePrint Arch.*, 2016:263, 2016.

[BCMS20] Benedikt Bünz, A. Chiesa, P. Mishra, and N. Spooner. Proof-carrying data from accumulation schemes. *IACR Cryptol. ePrint Arch.*, 2020:499, 2020.

[BF91] L. Babai and L. Fortnow. Arithmetization: A new method in structural complexity theory. *computational complexity*, 1:41–66, 1991.

[BFL91] L. Babai, L. Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *computational complexity*, 1:3–40, 1991.

[BGH20] Sean Bowe, J. Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. 2020.

[BLNS20] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Kim Khanh Nguyen, and Gregor Seiler. A non-pcp approach to succinct quantum-safe zero-knowledge. In *IACR Cryptol. ePrint Arch.*, 2020.

[BOGKW88] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: how to remove intractability assumptions. In *Providing Sound Foundations for Cryptography*, 1988.

[BSBHR17] E. Ben-Sasson, I. Bentov, Y. Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *Electron. Colloquium Comput. Complex.*, 2017.

[BSBHR18] E. Ben-Sasson, I. Bentov, Y. Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.

[BSCG+14] E. Ben-Sasson, A. Chiesa, Christina Garman, M. Green, Ian Miers, Eran Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014.

[BSCR+18] E. Ben-Sasson, A. Chiesa, Michael Riabzev, N. Spooner, M. Virza, and N. Ward. Aurora: Transparent succinct arguments for r1cs. In *IACR Cryptol. ePrint Arch.*, 2018.

[BSCTV14a] E. Ben-Sasson, A. Chiesa, Eran Tromer, and M. Virza. Scalable zero knowledge via cycles of elliptic curves. *IACR Cryptol. ePrint Arch.*, 2014:595, 2014.

[BSCTV14b] E. Ben-Sasson, A. Chiesa, Eran Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security Symposium*, 2014.

[CCH+18] R. Canetti, Yilei Chen, Justin Holmgren, A. Lombardi, G. N. Rothblum, and R. Rothblum. Fiat-shamir from simpler assumptions. *IACR Cryptol. ePrint Arch.*, 2018:1004, 2018.

[CCH+19] R. Canetti, Yilei Chen, Justin Holmgren, A. Lombardi, G. N. Rothblum, R. Rothblum, and Daniel Wichs. Fiat-shamir: from practice to theory. *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019.

[CLW19] R. Canetti, A. Lombardi, and Daniel Wichs. Fiat-shamir : From practice to theory , part ii nizk and correlation intractability from circular-secure fhe. 2019.

[COS19] A. Chiesa, Dev Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *IACR Cryptol. ePrint Arch.*, 2019.

[Dam10] I. Damgård. On sigma-protocols. 2010.

[Din06] I. Dinur. The pcp theorem by gap amplification. In *STOC '06*, 2006.

[ESLL19] Muhammed F. Esgin, R. Steinfeld, J. K. Liu, and D. Liu. Lattice-based zero-knowledge proofs: New techniques for shorter and faster constructions and applications. In *IACR Cryptol. ePrint Arch.*, 2019.

[FGM+89] M. Fürer, Oded Goldreich, Y. Mansour, M. Sipser, and S. Zachos. On completeness and soundness in interactive proof systems. *Adv. Comput. Res.*, 5:429–442, 1989.

[FK94] U. Feige and J. Kilian. Two prover protocols: low error at affordable rates. In *STOC '94*, 1994.

[FL92] U. Feige and L. Lovász. Two-prover one-round proof systems: their power and their problems (extended abstract). In *STOC '92*, 1992.

[FLS90] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 308–317 vol.1, 1990.

[For87] L. Fortnow. The complexity of perfect zero-knowledge. In *STOC '87*, 1987.

[FRS88] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. In *Theor. Comput. Sci.*, 1988.

[FS90] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *STOC '90*, 1990.

[GGPR12] R. Gennaro, C. Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. *IACR Cryptol. ePrint Arch.*, 2012:215, 2012.

[GINX14] N. Gama, Malika Izabachène, P. Nguyen, and X. Xie. Structural lattice reduction: Generalized worst-case to average-case reductions and homomorphic cryptosystems. *IACR Cryptol. ePrint Arch.*, 2014:283, 2014.

[GK96] Oded Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25:169–192, 1996.

[GKR15] S. Goldwasser, Y. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62:27:1–27:64, 2015.

[GM17] Nicholas Genise and Daniele Micciancio. Faster gaussian sampling for trapdoor lattices with arbitrary modulus. In *IACR Cryptol. ePrint Arch.*, 2017.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *STOC '85*, 1985.

[GO90] Oded Goldreich and Y. Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7:1–32, 1990.

[Gol01] Oded Goldreich. Foundations of cryptography: Volume 1, basic tools. 2001.

[Gol02] Oded Goldreich. Zero-knowledge twenty years after its invention. *IACR Cryptol. ePrint Arch.*, 2002:186, 2002.

[Gol11] Oded Goldreich. Notes on levin's theory of average-case complexity. In *Studies in Complexity and Cryptography*, 2011.

[GS86] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. *Adv. Comput. Res.*, 5:73–90, 1986.

[GSV98] Oded Goldreich, A. Sahai, and S. Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In *STOC '98*, 1998.

[JNV+20] Zheng-Feng Ji, Anand Natarajan, T. Vidick, J. Wright, and H. Yuen. Mip*=re. *ArXiv*, abs/2001.04383, 2020.

[KW93] Mauricio Karchmer and A. Wigderson. On span programs. *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 102–111, 1993.

[LFKN92] Carsten Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39:859–868, 1992.

[MP11] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. *IACR Cryptol. ePrint Arch.*, 2011:501, 2011.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of sis and lwe with small parameters. *IACR Cryptol. ePrint Arch.*, 2013:69, 2013.

[MR08] Daniele Micciancio and Oded Regev. Lattice-based cryptography. 2008.

[Pap84] C. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31:288–301, 1984.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for np from (plain) learning with errors. In *IACR Cryptol. ePrint Arch.*, 2019.

[Sch80] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27:701–717, 1980.

[Set19] S. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *IACR Cryptol. ePrint Arch.*, 2019.

[Sha92] A. Shamir. Ip = pspace. *J. ACM*, 39:869–877, 1992.

[Val79] L. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC*, 2008.

[WSR+15] Riad S. Wahby, S. Setty, Zuocheng Ren, A. Blumberg, and M. Walfish. Efficient ram and control flow in verifiable outsourced computation. *IACR Cryptol. ePrint Arch.*, 2014:674, 2015.

[WTS+18] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 926–943, 2018.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Y. Zhang, Charalampos Papamanthou, and D. Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *IACR Cryptol. ePrint Arch.*, 2019.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, 1979.

Appendix A

Indistinguishability

We explore the definitions of indistinguishability, in particular the indistinguishability of ensemble distributions. The word 'ensemble' is intended to capture how the distributions are viewed together rather than individually, in the same way that a musical ensemble consists of musicians playing together rather than individually. There are three types of ensemble indistinguishability. The strongest type is perfectly indistinguishable ensembles. The second type, also considered strong, is statistically indistinguishable ensembles. The weakest type, dependent on intractability assumptions, is computationally indistinguishable ensembles.

We begin by briefly introducing probability spaces and random variables. Here we only account for discrete probably distributions, that is where the number of possible outcomes is countable, whether finitely countable or infinitely countable. Then we introduce a statistical distance for measuring the similarity of distributions. Extending the notion of statistical distance to ensembles, we derive the three types of definitions for indistinguishable ensembles.

A.1 Probability definitions

A (discrete) **probability space**, more often called a **probability distribution** or just a **distribution**, is a (countable) set of possible outcomes Ω called the **sample space** paired with a function $p\colon \Omega \to [0,1]$ called the **probability function**. The probability function assigns a probability to each output, and all probabilities sum to one. Any subset $E \in \Omega$ is called an **event**, and the probability of the event occurring is $\sum_{e \in E} p(e)$. Often we abuse notation and denote the probability of the event by $p(E)$ or $\Pr[\text{describe}(E)]$ where describe(E) is some informal specification of the subset E and the probability function p is left implicit. Thus all probabilities summing to one may be written as $p(\Omega) = 1$ or $\Pr[\text{something occurs}] = 1$. In the discrete case the probability function can be called a **probability mass function** or **distribution function** to reflect how it assigns fractional probabilities across a sample space, analogous to how a physical distribution function assigns fractional weights across a physical space.

A (discrete) **random variable** $R\colon A \to B$ is a function between sample spaces, or between distributions when the sample spaces are appropriately paired with probability functions. Given a probability function p_A on A, the random variable R induces a probability function p_B on B. Specifically, we may use the natural definition $p_B(b) = \sum_{a \in R^{-1}(b)} p_A(a)$.

Thus when the domain of a random variable is a distribution, the random variable turns the codomain into a distribution. Since a random variable gives rise to a new distribution, the random variable is naturally associated with the new distribution. Notation is often abused by identifying the random variable as the new distribution itself, for example by identifying R as the distribution with sample space B with probability function p_B. So often the words 'distribution' and 'random variable' are used synonymously when actually random variables are functions between distributions.

Remark 4 (Differences from the measure theoretic definition)**.** *Typically a probability space is a triple consisting of a sample space, probability function, and additionally a set of sample space subsets called a σ-algebra, or an . These subsets are needed because the probability function may not be definable on the non-discrete sample spaces but only on certain subsets of the sample space. In the discrete case, however, the probability function may be defined on the sample space directly forgoing the σ-algebra.*

Next we define ensembles in the context of probability. Often in complexity theory or cryptography, we wish to parameterize a random variable by a string or a number such as a security parameter. Such parameterization is usually done using ensembles consisting of an infinite sequence of random variables with each random variable corresponding to a parameter. In our context the random variable usually ranges of strings of a certain size relative to the parameter. If the parameter is a number or a string, the random variable ranges over strings of length polynomial in the number or the length of the string respectively.

Definition 23 (Distribution ensembles)**.** *A **distribution ensemble** (more names mentioned below) is a countably-infinite sequence of distributions or random variables presumably related in some way. The notation is usually written as $\{X_i\}_{i\in I}$ or $\{X(i)\}_{i\in I}$ for some countable set I with some implicit ordering on I specifying the sequence. Such a set I is called an **indexing set**. Often I is taken to be the set of natural numbers $\mathbb{N}$, and indeed all countable sets are isomorphic to $\mathbb{N}$. Thought of another way, an ensemble is a stochastic process in which the indexing set is generalized beyond a time sequence.*

*Alternatives names include **probability ensemble**, **random variable ensemble**, or sometimes when clear from context just **ensemble**.*

Since distribution ensembles are a natural generalization of distributions, we may naturally extend random variables from mapping between distributions to mapping between distribution ensembles. Usually the distributions in a distribution ensemble are all related in a natural way such that a single random variable can be applied to all of them. For example, we will see below that an algorithm can thought of as a random variable applied to a distribution ensemble in which each distribution ranges over problems of a certain size. Given a distribution ensemble $\{X_n\}_{n\in\mathbb{N}}$ and a random variable R that may be applied to each distribution, we may create the new distribution ensemble $\{R(X_n)\}_{n\in\mathbb{N}}$. In the case that a single random variable cannot be applied to all distributions, one may instead apply a random variable ensemble $\{R_n\}_{n\in\mathbb{N}}$ resulting in distribution ensemble $\{R_n(X_n)\}_{n\in\mathbb{N}}$. For example, such is the case if R is a non-uniform algorithm.

A.2 Distribution Similarity

Given a sample space S, multiple probability functions may be defined on S, say p_1 and p_2, giving rise to multiple probability distributions. Since the distributions are defined on the same space, we may naturally compare them via the relative probabilities they assign to each outcome in the space. Of course, the two distributions are identical if and only if they assign the same probability to each outcome. In this case we say the statistical distance between them is zero. We now present the definition of statistical distance as a means for comparing the similarity of distributions. There are many definitions of statistical distance, but we focus on a certain one traditionally called 'total variation distance' which is natural for probability distributions.

Definition 24 (Statistical distance between distributions)**.** *Suppose we are given two distributions X and Y defined on the same sample space S, with corresponding probability functions p_X and p_Y.*

*The **statistical distance** between X and Y is defined as follows.*

$$\Delta(X, Y) \coloneqq \max_{E \subseteq S} |p_X(E) - p_Y(E)|$$

For each event E the distributions assign respective probabilities $p_X(E)$ and $p_Y(E)$. The statistical difference is the maximum difference the two distributions assign to the same event.

Proposition 5 (An equivalent definition)**.** *An equivalent definition for the case of discrete distributions is as follows.*

$$\Delta(X, Y) \coloneqq |p_X(S) - p_Y(S)|/2$$

This definition suffices for our purposes.

Proof. The event $E' :- \{s \in S | p_X(s) \geq p_Y(s)\}$ is one of the events upon which p_X and p_Y show maximum difference, that is $\Delta(X, Y) = |p_X(E') - p_Y(E')|$. To see this, note that for any other event E'' we have $p_X(E'' \setminus E') < p_Y(E'' \setminus E')$ by definition of E'. Therefore

$$\begin{aligned} |p_X(E') - p_Y(E')| &\geq |(p_X(E') - p_Y(E')) + (p_X(E'' \setminus E') - p_Y(E'' \setminus E'))| \\ &= |p_X(E'') - p_Y(E'')| \end{aligned}$$

We continue by noting the following equivalence.

$$\begin{aligned} & p_X(S) = 1 = p_Y(S) \\ \implies & p_X(E') + p_X(S \setminus E') = p_Y(E') + p_Y(S \setminus E') \\ \implies & p_X(E') - p_Y(E') = p_Y(S \setminus E') - p_X(S \setminus E') \end{aligned}$$

We now use the equivalence in the second equality below to transform the alternative definition into the original definition.

$$\begin{aligned} & |p_X(S) - p_Y(S)|/2 \\ &= (p_X(E') - p_Y(E'))/2 + (p_Y(S \setminus E') - p_X(S \setminus E'))/2 \\ &= p_X(E') - p_Y(E') = |p_X(E') - p_Y(E')| \\ &= \max_{E \subseteq S} |p_X(E) - p_Y(E)| \end{aligned}$$

A.3 Indistinguishable ensembles

First we state the traditional definition of computational indistinguishability and then we present our own definition in terms of statistical distance and finally show the two definitions are equivalent. The traditional definitions of perfect and statistical indistinguishability are identical to ours.

Definition 25 (The typical definition)**.** *Two distribution ensembles* $\{X_n\}_{n\in N}$ *are* $\{Y_n\}_{n\in N}$ *are* ***computationally indistinguishable*** *if for every p.p.t.distinguisher* D *the following is a negligible function in* n*.*

$$|\Pr D(X_n) = 1 - \Pr D(Y_n) = 1|$$

The probabilities are taken not just over the sampling of either X_n *or* Y_n *but also over the internal coin tosses of* D*.*

We first briefly describe our three forms of indistinguishable distributions, and then we will extend to indistinguishable distribution ensembles. Two distributions are **perfectly indistinguishable** if they are identical. Two distributions X and Y are **statistically indistinguishable** with respect to security parameter λ if their statistical distance is negligible, that is $\Delta(X, Y) \leq 1/2^\lambda$. Now regarding computational indistinguishability with respect to security parameter λ, suppose X and Y are two distributions defined on the same sample space S, and U is the uniform distribution on $\{0,1\}^{\text{poly}}\lambda$. The distributions X and Y are **computationally indistinguishable** if for every polynomial-time computable random variable $D\colon S \times U \to \{0,1\}$ we have $\Delta(D(X,U), D(Y,U)) \leq 1/2^\lambda$. In order to interpret D as a distinguisher we restrict it to a binary output set such that we may interpret its output as pointing to either X or Y.

We now describe in more detail what we mean by an algorithm as a random variable, particularly in the context of ensembles. Whereas above we explicitly stated a security parameter λ, here the security parameter is implicitly taken to be the size of the problems. Suppose a randomized algorithm A is defined on the problem sets $\{0,1\}^n$ for $n \in N$ and when receiving a problem of size n it receives a random auxiliary input from $\{0,1\}^{\text{poly}}n$. Therefore we may think of A as a random variable defined on the sequence of sample spaces $\{\{0,1\}^n \times \{0,1\}^{\text{poly}}n\}_{n\in\mathbb{N}}$. Or rather, A is a random variable defined on the distribution ensemble $\{\{0,1\}^n \times \{0,1\}^{\text{poly}}n\}_{n\in\mathbb{N}}$ in which each set $\{0,1\}^n$ is subject to some probability function for sampling problems, and each set $\{0,1\}^{\text{poly}}n$ is subject to the uniform probability function. A p.p.t.algorithm A is the special case that the random variable is polynomial-time computable, meaning there exists a deterministic algorithm that given any input $(x, r) \in \{0,1\}^n \times \{0,1\}^{\text{poly}}n$ computes $A(x, r)$ in time polynomial in $|(x, r)|$. We will treat a p.p.t.distinguisher as such a random variable in our definition of computationally indistinguishable ensembles below.

Now we state our three definitions of ensemble indistinguishability.

Definition 26 (Perfectly indistinguishable ensembles)**.** *Two distribution ensembles* $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *are* ***perfectly indistinguishable*** *if they are identical, that is* $X_n = Y_n$ *for all* n*.*

Definition 27 (Statistically indistinguishable ensembles)**.** *Two distribution ensembles* $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *are* ***statistically indistinguishable*** *if the statistical distance between pairs of random variables is negligible for sufficiently large* n*. In other words,* $\Delta(X_n, Y_n)$ *is a negligible function of* n*.*

Definition 28 (Computationally indistinguishable ensembles)**.** *Let* $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *be two distribution ensembles defined on the same sequence of sample spaces* $\{S_n\}_{n\in\mathbb{N}}$*. Let* $\{U_n\}_{n\in\mathbb{N}}$ *be the distribution ensemble in which each* U_n *is the uniform distribution over* $\{0,1\}^{\text{poly}}n$*. We may then create the sequence of sample spaces* $\{S_n \times \{0,1\}^{\text{poly}}n\}_{n\in N}$ *upon which we have the two distribution ensembles* $\{X_n \times U_n\}_{n\in N}$ *and* $\{Y_n \times U_n\}_{n\in N}$*.*

The ensembles $\{X_n\}_{n\in\mathbb{N}}$ *and* $\{Y_n\}_{n\in\mathbb{N}}$ *are* ***computationally indistinguishable*** *if for every polynomial-time computable random variable* $D\colon S_n \times \{0,1\}^{\text{poly}}n \to \{0,1\}$ *(mapping from sample spaces* $\{S_n \times \{0,1\}^{\text{poly}}n\}_{n\in\mathbb{N}}$ *to some binary set such as* $\{0,1\}$*) the following two distribution ensembles are statistically indistinguishable.*

$$\{D(X_n, U_{\text{poly}}n)\}_{n\in\mathbb{N}} \text{ and } \{D(Y_n, U_{\text{poly}}n)\}_{n\in\mathbb{N}}$$

Now we show the equivalence between the traditional definition and our latter definition.

Proposition 6 (Equivalence of the two definitions)**.** *The traditional definition of computationally indistinguishable ensembles is equivalent to the definition directly above.*

Proof. It is sufficient to show that $\Delta(D(X_n, U_{\text{poly}}n), D(Y_n, U_{\text{poly}}n)$ is equivalent to $|\Pr D(X_n) = 1 - \Pr D(Y_n) = 1|$ because then one is a negligible function of n if only if the other is.

Let p_X be the probability function corresponding to distribution $X_n \times U_{\text{poly}}n$, and likewise p_Y corresponding to $Y_n \times U_{\text{poly}}n$.

$$\begin{aligned}
&\Delta(D(X_n, U_{\text{poly}}n), D(Y_n, U_{\text{poly}}n)) \\
&= \frac{1}{2}\sum_{b\in\{0,1\}} |p_X(D^{-1}(b)) - p_Y(D^{-1}(b))| \\
&= \frac{1}{2}(|(1 - p_X(D^{-1}(1))) - (1 - p_Y(D^{-1}(1)))| + |p_X(D^{-1}(1)) - p_Y(D^{-1}(1))|) \\
&= \frac{1}{2}(|p_Y(D^{-1}(1)) - p_X(D^{-1}(1))| + |p_X(D^{-1}(1)) - p_Y(D^{-1}(1))|) \\
&= |p_X(D^{-1}(1)) - p_Y(D^{-1}(1))| \\
&= |\Pr D(X_n, U_{\text{poly}}n) = 1 - \Pr D(X_n, U_{\text{poly}}n) = 1|
\end{aligned}$$

The last expression differs from the traditional definition in that the p.p.t.distinguisher D is randomized via a random auxiliary input from $U_{\text{poly}}n$ rather than from internal coin tosses. Yet these two forms of randomization are equivalent, completing the equivalence.

The above definitions refer to computational indistinguishability when the distinguisher only has a single sample to examine. If the distinguisher is able to generate many samples of both distributions on its own, or receive them from some oracle, indistinguishability under a single sample implies indistinguishability under many samples. We do not prove this here, but provide an idea how we may argue reducibility using what's called a 'hybrid argument.'

For $m = \text{poly}n$ consider the $m+1$ distribution ensembles $\{(X_n^{(1)}, \ldots, X_n^{(k)}, Y_n^{(k+1)}, \ldots, Y_n^{(m)})\}_{n\in\mathbb{N}}$ for $0 \leq k \leq m$. Ensemble $k = 0$ is $\{(X_n^{(1)}, \ldots, X_n^{(m)})\}_{n\in\mathbb{N}}$ and ensemble $k = m$ is $\{(Y_n^{(1)}, \ldots, Y_n^{(m)})\}_{n\in\mathbb{N}}$. If one can distinguish between ensembles 0 and m, and one can construct all intermediate ensembles, then there must exist an index $0 \leq k < m$ such that one can distinguish between neighboring ensembles k and $k+1$. Distinguishing between neighboring ensembles k and $k+1$ entails distinguishing between each of their distributions, including distribution $k+1$ which is either $Y_n^{(k+1)}$ or $X_n^{(k+1)}$ respectively. Thus the ability to distinguish under $m = \text{poly}n$ samples implies the ability to distinguish under 1 sample. The contrapositive means indistinguishability under one sample implies indistinguishability under poly$n$ samples. See a proof in [Gol01] page 108.

Appendix B

Selecting parameters for SIS

B.1 Introduction

In our lattice based construction 5.2 of saga batching schemes, use short integer solution problem (SIS) to take advantage of its homomorphic properties.

Concrete parameters are irrelevant to the theoretical properties of proof systems employing SIS. Such properties are the asymptotic time and space complexities for both prover and verifier. But the concrete parameters are absolutely relevant for the practical complexities of proof systems employing SIS. The larger these parameter constants, the less efficient the implementation is in practice, to the point when an implementation is simply no longer practical.

Below we specify the formal definition of the SIS problem, and the subsequenly explore theoretical parameters and then concrete parameters.

B.2 The SIS Problem

Definition 29 (SIS Problem)**.** *Let modulus q, dimensions m and n, norm-parameter p, and bound β be natural numbers. Given a uniformly random matrix $A \in \mathbb{Z}_q^{m \times n}$, the* Short Integer Solution (SIS) *problem is to find a witness $x \in \mathbb{Z}^n$ such that $x \neq 0$, $\|x\|_p \leq \beta$, and $Ax = 0 \mod q$.*

One may interpret A as defining a random q-ary lattice of dimension n, with lattice points as integer vectors orthogonal to all m rows. Then the problem is to find a lattice point with p-norm no more than β.

The SIS problem was proven in 1996 by Ajtai's work [Ajt96] to be as difficult as worst-case lattice problems. Ajtai's work is seminal and particularly celebrated when it comes to post-quantum cryptography. The reason this reduction from SIS to worst-case lattice problems is so significant is that existing cryptographyic assumptions, such as the discrete-log and integer-factorization problems, are not well founded by worst-case assumptions. Rather, existing assumptions are founded by the inability of experts to find successful solutions over the decades. Of course, for the worst-case lattice assumptions upon which we now depend to mean anything, they also must lack successful assumptions by experts over the years. This

is indeed the case. While both existing cryptographic assumptions and worst-case lattice assumptions rest upon time-tested lack of success, worst-case assumptions are far stronger complexity assumptions than average-case assumptions, and are thus preferable.

We are interested in selecting parameters for high density SIS instances. High density is when $m \log_2(q) \ll n$, or more generally when input is significantly larger than output. A line of work has followed, improving the parameters and the tightness of the reduction. The reductions employed set m as the dimension of the worst-case lattice, and $q, n = \text{poly}(m)$. The p-norm is usually taken to be the Euclidean norm for which $p = 2$. The bound β must be large enough that a solution exists, but small enough that the problem is non-trivial. A loose inequality requires $m \log(q) < \beta < q$.

B.3 Theoretical Parameters

The tightest theoretical reduction published thus far from SIS to worst-case lattice problems comes from [MP13] which provides nearly optimal asymptotic parameters. In particular, setting m as the security parameter and norm-paramter $p = 2$ (Euclidean norm), Micciancio and Peikert show that for any $\epsilon > 0$ an efficient reduction exists as long as $q \geq \beta \cdot n^{\epsilon}$. Note that this result is independent of n. This result improves upon an iterative line of work dating back to Ajtai's original reduction [Ajt96].

Papers such as [ESLL19] that implement SIS select parameters by following [MR08], which analyzes the cost of the random (q-ary) n-dimensional lattice problem, ignoring the reduction to, and cost of the respective worst-case m-dimensional lattice problem. This indicates that in practice average-to-worst-case reductions are only of theoretical, asymptotic interest and not relevant when selecting practical, concrete parameters. The sole value of average-to-worst-case reductions a solid reassurance that attack algorithms will not significantly improve in the future. Therefore we follow [MR08] below in an effort to select safe parameters for high density SIS, in order to construct a highly compressive collision-resistant hash functions. We consider the infinite norm with bound $d < q$.

B.4 Avoiding lattice reduction attacks

In reasonable time one may select any subset of c columns of A and find a solution x with $\|x\|_2 \geq q^{n/c}\delta^c$ by setting coordinates for ignored columns to 0. In this case δ is called the "root Hermite factor". An x with $\|x\|_\infty < d$ will have $\|x\|_2 \leq \sqrt{c(d-1)^2}$. Thus it is sufficient to set the constraint $\forall c \in [n] : q^{m/c}\delta^c > (d-1)\sqrt{c}$. This can be modified to $\forall c \in [n] : m \log_2(q) > c \log_2((d-1)\sqrt{c}/\delta^c)$.

B.5 Avoiding combinatorial attacks

For the smallest integer k satisfying

$$\frac{2^k}{k+1} \geq \frac{n \log_2(2(d-1)+1)}{m \log_2(q)}$$

one can find solution x with $\|x\|_\infty < d$ in time $(2(d-1)+1)^{n/2^k}$. Thus for security parameter s we add the constraint $(2(d-1)+1)^{n/2^k} \geq 2^s$.

B.6 An ambitious selection

As far as we know the above constraints are sufficient. As an example, we show a very ambitious example of parameters, achieving the major hash compression factor 6000) as follows.

$$\begin{aligned} d &= 64 \\ n &= 2 \times 10^6 \\ m \log_2(q) &= 2 \times 10^3 \end{aligned}$$

For $\delta = 1.01$ (could be reduced for quantum algorithms) the maximum of $c \log_2((d-1)\sqrt{c}/\delta^c)$ is about 1800 at about $c = 400$. Thus the first constraint is satisfied with

$$m \log_2(q) = 2000 > 1800 = \max_c \{c \log_2((d-1)\sqrt{c}/\delta^c)\}$$

For $k = 17$

$$\frac{2^k}{k+1} = 7281 \geq 6989 = \frac{n \log_2(2(d-1)+1)}{m \log_2(q)}$$

So the cost is $(2(d-1)+1)^{n/2^k} = 2^{n \log_2(2(d-1)+1)/2^k} = 2^{106}$ offering 106 bits of security. Note the precise values of m and q are left free subject to $m \log_2(q) = 2 \times 10^3$ and $q > 64$.