

zkQMC: Zero-Knowledge Proofs For (Some) Probabilistic Computations Using Quasi-Randomness

Zachary DeStefano
zdestefano@lanl.gov

Dani Barrack
dbarrack@lanl.gov

Michael Dixon
mdixon@lanl.gov

Los Alamos National Laboratory

Abstract

We initiate research into efficiently embedding probabilistic computations in probabilistic proofs by introducing techniques for capturing Monte Carlo methods and Las Vegas algorithms in zero knowledge and exploring several potential applications of these techniques. We design and demonstrate a technique for proving the integrity of certain randomized computations, such as uncertainty quantification methods, in non-interactive zero knowledge (NIZK) by replacing conventional randomness with low-discrepancy sequences. This technique, known as the *Quasi-Monte Carlo* (QMC) method, functions as a form of weak algorithmic derandomization to efficiently produce adversarial-resistant worst-case uncertainty bounds for the results of Monte Carlo simulations. The adversarial resistance provided by this approach allows the integrity of results to be verifiable both in interactive and non-interactive zero knowledge without the need for additional statistical or cryptographic assumptions.

To test these techniques, we design a custom domain specific language and implement an associated compiler toolchain that builds zkSNARK gadgets for expressing QMC methods. We demonstrate the power of this technique by using this framework to benchmark zkSNARKs for various examples in statistics and physics. Using N samples, our framework produces zkSNARKs for numerical integration problems of dimension d with $O\left(\frac{(\log N)^d}{N}\right)$ worst-case error bounds. Additionally, we prove a new result using discrepancy theory to efficiently and soundly estimate the output of computations with uncertain data with an $O\left(d\frac{\log N}{\sqrt{N}}\right)$ worst-case error bound. Finally, we show how this work can be applied more generally to allow zero-knowledge proofs to capture a subset of decision problems in BPP, RP, and ZPP.

1 Introduction

Motivation. Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) [Bit+11] [GGPR] have been used to show that a deterministic computa-

tion was executed correctly or that a satisfying path exists through a non-deterministic computation [Par+13] [Ben+13]. They have been applied to a number of areas including image processing [Ko+20], finance [Ben+14b], optimization [Ang+21], and Machine Learning [Lee+20] [DeS20]. However, the process of designing adversarially resistant non-interactive zero-knowledge protocols (e.g. zkSNARKs) for computations which involve prover-generated randomness and uncertainty is not well understood, and current approaches require additional statistical and cryptographic assumptions or machinery.

Frequently in statistical physics [BH10] and finance [BBG97] [LL03] applications, the Monte Carlo method is used to approximate simulations and problems which do not admit closed-form solutions because it has excellent average-case performance and convergence. The Monte Carlo method requires sufficient quality randomness to be provided as an input in order to guarantee both average-case convergence and the soundness of the confidence interval estimation. In a non-interactive adversarial setting, there are several challenges to the verifiable uncertainty quantification of the Monte Carlo method.

Efficiently circumventing the statistical and cryptographic assumptions (and additional security concerns) currently required when designing an adversarially resistant non-interactive zero-knowledge protocol attesting to the results of a randomized computation, such as the Monte Carlo method, would enable new capabilities and bring multiple benefits. First, some fully deterministic zkSNARK gadgets could be replaced with more efficient derandomized variants of randomized gadgets, thereby improving prover performance agnostic of zkSNARK backend choice. Second, zkSNARKs would be able to efficiently capture knowledge about quantifiably uncertain facts or approximation results. For example, in addition to proving the correct execution of a program which approximates an uncertain value, we could provide additional meaningful guarantees that the approximation is within a certain distance of the true value. We discuss several approaches to this problem and these challenges in more detail in Section 3.

Contribution. To overcome these challenges and achieve verifiable uncertainty quantification of the Monte Carlo (MC) method in non-interactive zero knowledge without additional statistical or cryptographic assumptions, we consider the use of Quasi-Monte Carlo (QMC)[Nie92] [Caf98] methods as a lightweight form of derandomization where randomness is replaced by a *deterministic*, low-discrepancy (quasi-random) sequence. This technique provides unconditional bounds on the worst-case error of the result of Monte Carlo method computations, even in the presence of an adversarial prover. To demonstrate its benefits, we compare this Quasi-Monte Carlo method to several alternative approaches for proving statements using the Monte Carlo method in non-interactive zero knowledge with various cryptographic, statistical, and computational assumptions.

We apply and extend QMC for use with non-interactive zero-knowledge proofs, such as zkSNARKs, develop a framework for applying QMC in zero knowledge, and benchmark the overhead of our implementation for three example problems of increasing complexity from statistics and physics. Additionally, we describe how these worst-case error bounds and accompanying proofs can be reused as inputs for more complex systems which just leverage the Monte Carlo method as a subroutine. To this end, we provide an algorithm for computing worst-case error bounds on the evaluation of $f(u)$ when u is partially known. Given a known d -dimensional interval, I , s.t. $u \in I$, we show that the application of our technique can efficiently compute $f(u)$ with $O\left(d \frac{\log N}{\sqrt[4]{N}}\right)$ precision in non-interactive zero knowledge. Using this result, we extend the analysis of our system to include situations involving mutually distrustful provers ($\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, etc.) where the interval results of Monte Carlo method subroutines can be used as inputs to later Monte Carlo calculations without unnecessary loss of soundness or accuracy. Going beyond Monte Carlo methods to different types of randomized computations, we also provide details on how to efficiently embed Las Vegas algorithms in zero knowledge in Appendix A.

2 Background

Monte Carlo and Quasi-Monte Carlo. The Monte Carlo (MC) method is a class of techniques for using randomness to approximate functions which are too computationally complex to calculate analytically [MU49] [Met+53] [Caf98]. Often these functions take the form of simulations or physical processes involving risk and uncertainty [Cun+14] [Zha21], making the Monte Carlo method a powerful technique used in modern decision-making and design [Kro+14].

The **Monte Carlo (MC)** method is a direct consequence of the application of the central limit theorem

(and law of large numbers) to integral approximation via random sampling. Given a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ with bounded variation (finite $\text{Var}[f(X)]$), the goal is to estimate $\int_{[0,1]^d} f(u)du$ using the average of N independent random variables X_1, \dots, X_N : $\frac{1}{N} \sum_{i=1}^N f(X_i)$. This discrete sum is an unbiased estimator with well defined variation:

$$\mathbb{E} \left[\frac{1}{N} \sum_{i=1}^N f(X_i) - \int_{[0,1]^d} f(u)du \right] = 0$$

$$\text{Var} \left[\frac{1}{N} \sum_{i=1}^N f(X_i) - \int_{[0,1]^d} f(u)du \right] = \frac{\text{Var}[f(X)]}{N}.$$

Given bounded variation, the average-case error for the Monte Carlo method is $O(N^{-1/2})$ which provides a baseline average convergence independent of dimension; however, the worst-case error is independent of the number of random variables. We discuss this error, which is directly related to the variation of the function, in Section 3. By direct application of the central limit theorem, the Monte Carlo method can be used to produce confidence intervals approximating the true value of $\int_{[0,1]^d} f(u)du$ to any degree of statistical soundness; however, to bound the worst-case error, we need to instead apply the Quasi-Monte Carlo method.

The **Quasi-Monte Carlo (QMC)** method is a modification of the traditional Monte Carlo method, frequently used in finance [CHL15] [HK21] and computer graphics [SEB08], where the N independent random variables, X_1, \dots, X_N , are replaced by a deterministic sequence, x_1, \dots, x_N , with *low discrepancy* over the interval $[0, 1]^d$. We provide a more rigorous definition of discrepancy and key results in the context of sequences on intervals in the next subsection and show several common examples, but as intuition, a sequence is said to have *low discrepancy* over a particular interval if it uniformly covers that interval without clumps or large gaps.

Using this approach, one can bound the worst-case error by $O\left(\frac{(\log N)^d}{N}\right)$ [Nie92]. This is accomplished by using the Koksma–Hlawka inequality [Hla61] [Ais+16] and sequences with (conjectured) minimal discrepancy [Nie92]. Specifically the QMC method applied in this way gives the following worst-case error bound on our estimation:

$$\left| \frac{1}{N} \sum_{i=0}^N f(x_i) - \int_{[0,1]^d} f(u)du \right| \leq C \frac{(\log N)^d}{N} \text{Var}_{HK}(f)$$

for some constant C , where $\text{Var}_{HK}(f)$ is the Hardy–Krause variation of the function f [Owe04].

Average-Case Worst-Case Tradeoffs. Comparing the worst-case error bound provided here by QMC and the nice average-case error provided by MC highlights

a key fundamental distinction between these two approaches. There is a direct relationship between the discrepancy of the point set used for approximation and the magnitude of the resulting worst-case error which provides a natural intuition for why using a random sequence of points does not provide non-trivial worst-case error guarantees. A random sequence of points is not necessarily low-discrepancy, that is to say it does not need to evenly cover an interval, but instead it can form clumps and large gaps. While the QMC method is superior to the MC method in respect to providing stronger guarantees, namely a concrete worst-case error bound, it does not provide a nice measure of average-case convergence.

There is a natural objection here, namely that while using the MC method, we could simply compute the discrepancy of the random sequence chosen and produce worst-case error bounds while maintaining well defined average-case convergence using this calculated value. While this is certainly possible, this problem of calculating this discrepancy (technically a specific type called *star discrepancy* which we properly introduce in the next section) is known to be NP-hard in the case where $N \approx d$ [GSW09]. When $N \gg d$, the best known algorithm for computing the discrepancy of an arbitrary sequence runs in time $O(N^{1+d/2})$ [DEM96] with approximations being similarly expensive [DGW14].

The structure of low-discrepancy sequences is directly responsible for the lack of non-trivial average-case error convergence for QMC. The structure inherent in these sequences leads to the inapplicability of the central limit theorem to the QMC method and provides a natural motivation for why the problem of confidence interval estimation is hard in this setting. Empirically, there are results which suggest that *hybrid-QMC*, QMC where the low-discrepancy sequence is shifted by a random seed, has fast average-case approximation convergence; however, this heavily depends on the behavior of the function being integrated [Pap03] [Tuf04]. We briefly touch on some of these average-case error results again in our discussion of future work.

The Discrepancy of Sequences. The discrepancy and star discrepancy, D_N and D_N^* , of a sequence of N points P in the context of the unit hypercube $[0, 1]^d$ are respectively defined using Niederreiter's [Nie92] notation as

$$D_N(P) = \sup_{B \in \mathcal{J}} \left| \frac{A(B; P)}{N} - \lambda_d(B) \right|$$

$$D_N^*(P) = \sup_{B \in \mathcal{J}^*} \left| \frac{A(B; P)}{N} - \lambda_d(B) \right|$$

where $A(B; P)$ denotes the number of points from P in B , $\lambda_d(B)$ is the d -dimensional Lebesgue measure of B , \mathcal{J} is the set of all subintervals of $[0, 1]^d$, and \mathcal{J}^* is the set of all subintervals of $[0, 1]^d$ anchored on the origin.

The discrepancy and star discrepancy are related by the following inequality [Nie92],

$$D_N^*(P) \leq D_N(P) \leq 2^d D_N^*(P).$$

Niederreiter conjectures that $B_d \frac{(\log N)^d}{N}$ is the lowest star discrepancy that a sequence can achieve where B_d is some small constant determined by the dimension d .

There are several major families of low-discrepancy sequences. The first of these families is the Kronecker / Ramshaw / Richtmyer / Weyl sequences [LN93] [Ram81] [Ric51] [Wey16]. Referenced by several different names, the standard form of an element of a sequence of this type is

$$x_k = (\{\gamma_1 k + c_1\}, \dots, \{\gamma_d k + c_d\})$$

where c_1, \dots, c_d are arbitrary constants, and $\gamma_1, \dots, \gamma_d$ are irrational with the condition that all γ_i and γ_j are linearly independent over \mathbb{Q} . The sequence is formed by fixing c_1, \dots, c_d and $\gamma_1, \dots, \gamma_d$ and taking all points x_k with $1 \leq k \leq N$. When $\gamma_1, \dots, \gamma_d$ are badly approximated by the rationals, this sequence has excellent low discrepancy, and it works for any choice of $\gamma_1, \dots, \gamma_d$ which satisfies our constraints [Bec94]. For the purposes of this paper, we will exclusively refer to this family of sequences as Kronecker sequences.

This family of sequences in particular has several nice properties which make them amenable for our context.

- They most avoid issues with non-uniform behavior which is present in some other low-discrepancy constructions, particularly at higher dimensions.
- The discrepancy is preserved when we consider these points on a torus (where \mathcal{J} can be a interval that wraps-around the edges of $[0, 1]^d$), so we can effectively arbitrarily pick c_1, \dots, c_d without changing the discrepancy.
- Individual elements of this sequence are easy to compute as a single element only requires d addition, multiplication, and modulus operations.

In addition to this construction, other common low-discrepancy sequences include Faure, Halton, Niederreiter, Sobol, and Van der Corput which use a variety of other techniques. A more detailed discussion and comparison of some of these techniques and applications can be found here [Nie92] [KW97] [Wei00] [Owe03].

Figure 1 shows the difference between 100 pseudo-random and 100 quasi-random points in \mathbb{R}^2 . The quasi-random points are generated from a Kronecker sequence. Notice how the previously discussed differences become visually apparent in these diagrams. The pseudo-random points exhibit non-uniform properties, forming clumps and large gaps, while the quasi-random points exhibit more structure.

For the remainder of the paper, we exclusively use Kronecker sequences in our approach, applications, and

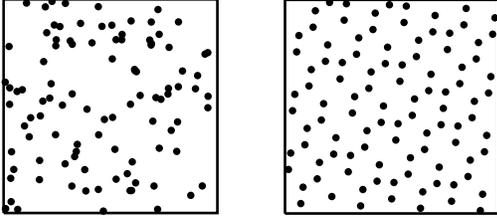


Figure 1: 100 Pseudo-Random points in \mathbb{R}^2 (left) and 100 Quasi-Random (Kronecker) points in \mathbb{R}^2 (right).

discussion; however, with judicious consideration of non-uniform low order factors, any of the other low-discrepancy sequences would provide the same guarantees up to a constant factor. Additionally, we use the terms quasi-random and low-discrepancy interchangeably when describing sequences.

Zero-Knowledge Proofs and zkSNARKs. This section is intended to provide sufficient background on zero-knowledge proofs, a specific type of zero-knowledge proof called a zkSNARK, and the specific zkSNARK protocol we used for our implementation. While our implementation specifically targets R1CS-based zkSNARK architectures, the technique we introduce generalizes to other non-interactive proof protocols. Additionally, following the nomenclature used in zero-knowledge proof literature, we use *argument of knowledge* and *proof* interchangeably because arguments of knowledge are computationally sound proofs.

A **Zero-Knowledge Proof**, introduced in [GMR85], is a probabilistic protocol where a prover \mathcal{P} convinces a skeptical recipient (also called a verifier \mathcal{V}) with high probability that some statement is true without revealing any additional information in the process. To prove a statement in zero knowledge is precisely to convince a verifier with high probability that some statement is true without revealing any additional knowledge, and we denote this proof as π .

Combining this interactive protocol with modern cryptography allows for **Non-Interactive Zero-Knowledge Proofs** (NIZKs) [BFM88] which, under certain cryptographic assumptions, allow for a prover to prove statements in zero knowledge without any interaction with the verifier.

Further developments in this area enhanced the power of these proofs [GMW91] and minimized proof size [Kil92] [Mic00] which culminated in **Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge** (zkSNARKs) [Groth10] [Bit+11] [GGPR].

Since the introduction of the first zkSNARKs construction, many other non-interactive zero-knowledge proof systems have been designed with a variety of attributes and strengths including [Wah+17] [Set20] [BGH19] [Mal+19], and our work is general enough that it applies to all of these. For our implementa-

tion, we chose Libsnark [SCIPR] which implements the Groth16 [Groth16] protocol for producing and verifying zkSNARKs.

The Libsnark library takes as input a Rank-1 Constraint System (R1CS) \mathcal{C} over \mathbb{F}_p and creates the following 3 functions:

- $Generate(\mathcal{C}) \rightarrow (\text{pk}, \text{vk})$, which is required to run once to produce a public proving key, pk, and public verification key, vk;
- $Prove(\text{pk}, X, W) \rightarrow \pi$, which takes the public proving key, a public vector of field values X , and a private vector of field values W and produces a zkSNARK π ;
- $Verify(\text{vk}, (X, \pi)) \rightarrow \top \vee \perp$, which efficiently uses the verification key, vk, to check if the public input and zkSNARK pair (X, π) constitute a valid proof (with a negligible chance of error).

This particular cryptographic protocol has several additional desirable properties: the length of the proof, $|\pi|$, is a small constant independent of circuit size, and *Prove* steps can be composed using the proof-carrying data paradigm [CT10] [Ben+14a] [CTV15]. The *Verify* step is cheap, and proof-carrying data allows for one prover \mathcal{P} to take several existing proofs (π_1, π_2, π_3 , etc.) in the *Prove* step and produce a single proof π which attests to the correctness of all of them. Both of these properties are desirable for our application but neither is strictly necessary.

3 Approach

We consider the problem of a prover \mathcal{P} who wants to produce a proof of some statement that can be efficiently probabilistically approximated using the Monte Carlo method. This proof should preserve soundness, avoid leaking any additional information about the statement, not require any additional statistical or cryptographic assumptions, and allow for efficient verification by any verifier \mathcal{V} .

We first briefly discuss the limitations of several designs for attempting to implement the Monte Carlo method in zero knowledge and how they fail on one or more of the aforementioned criteria. Then we discuss our approach using low-discrepancy sequences and show how it efficiently satisfies the desired criteria while avoiding the pitfalls of these other more complex designs.

Later we show how our approach can be extended to more complex systems involving several mutually distrustful provers ($\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, etc.) in Section 6.

Additionally we expand our scope beyond proofs of the Monte Carlo method to a different class of randomized computations, Las Vegas algorithms, in Appendix A.

Intuition. Intuitively, we can view the task of implementing the Monte Carlo method in zero knowledge as balancing the capabilities of two different adversaries. The first being an adversarial prover, who is incentivized to find the worst possible allowed sequence of inputs in order to detrimentally impact the soundness of the protocol. The second is the verifier, who is inclined to demand additional constraints on the sequences the prover can use which in turn allows for the extraction of additional information from the Monte Carlo method. We show that the proper use of low-discrepancy sequences satisfies this tension and thwarts both adversaries in a way that other approaches do not. These sequences are just structured enough to maintain soundness while providing the prover with sufficient freedom to maintain zero knowledge.

Limitations of the Monte Carlo Method. The primary challenge of proving statements about the Monte Carlo method comes from the random variables present in the algorithm. Approaches to enforcing the use of random variables generally fall into one of two pitfalls, either they break soundness by allowing for the prover \mathcal{P} to search over randomness, seeds, or time or they break the zero-knowledge requirement by leaking additional information about the full proof. We consider several different high level approaches here and discuss their flaws.

Suppose, trivially, that the prover \mathcal{P} is free to choose a sequence of points x_1, \dots, x_N . In this unconstrained case, zero knowledge is preserved, but soundness is clearly violated because an adversarial prover is under no obligation to pick these points from a claimed distribution.

Suppose then, that the prover \mathcal{P} is free to choose a seed point x_1 but is forced to use the sequence of points x_1, \dots, x_N generated pseudo-randomly from that seed. Let us also generously assume that this is an ideal cryptographic PRNG which produces a uniform distribution of points for a seed with high probability. In this case, computational zero knowledge is preserved but soundness is again threatened by the ability of a malicious prover to search over seed values.

If the prover \mathcal{P} is forced to use a seed point x_1 generated from a random oracle access, this regains the soundness of the protocol at the cost of breaking the zero-knowledge requirement ($x_1 \dots x_N$ becomes fully known to the verifier) and the introduction of a cryptographic assumption about random oracles. This analysis can be similarly extended to approaches that acquire randomness via some other cryptographic assumption. For example, using a randomness beacon introduces both a cryptographic assumption and the potential for an adversarial prover to search over time to acquire a sequence of points which misleads the verifier.

Suppose that we are willing to settle for a proof of

the Monte Carlo method where the prover has control over the seed, but the points following it are statistically random. By the central limit theorem, we should expect the results of the Monte Carlo method to obey a normal distribution, so, with certain assumptions on X_1, \dots, X_N , we have the following probability that the difference between our experimental result and the true result is within ε when we use N sample points and the variance of our results is σ (which would need to be estimated or bounded in some way)

$$\mathbb{P} \left[\left| \frac{1}{N} \sum_{i=1}^N f(X_i) - \int_{[0,1]^d} f(u) du \right| \leq \varepsilon \right] = 2\Phi \left(\frac{\varepsilon\sqrt{N}}{\sigma} \right) - 1$$

where Φ is the cumulative distribution function for the standard normal distribution.

Let $\alpha := 2\Phi \left(\frac{\varepsilon\sqrt{N}}{\sigma} \right) - 1$ be the probability that difference between our experimental result and the true result is within ε . This gives us three parameters α , ε , and N , so by fixing the first two, we can calculate the number of samples, N , required to satisfy

$$\mathbb{P} \left[\left| \frac{1}{N} \sum_{i=1}^N f(X_i) - \int_{[0,1]^d} f(u) du \right| \leq \varepsilon \right] = \alpha$$

for any combination of α and ε .

To shrink our confidence interval with probability α to width ε we need at least the following number of samples

$$N = \left(\frac{\Phi^{-1} \left(\frac{1+\alpha}{2} \right) \sigma}{\varepsilon} \right)^2.$$

If we assume that the prover can search over M independent sequences of random points and picks the worst one, then by union bound the number of samples N in a sequence required to attain the same confidence interval is

$$N = \left(\frac{\Phi^{-1} \left(\frac{1+\alpha^M}{2} \right) \sigma}{\varepsilon} \right)^2.$$

There is a significant overhead in the number of samples required for a prover to produce a proof that with probability α the result of the Monte Carlo method is within an interval of width ε . Furthermore this approach is fundamentally weaker than our approach using Quasi-Monte Carlo when Var_{HK} is bounded by some polynomial in d because, unlike with QMC, its soundness is directly a function of the number of samples.

Our Quasi-Monte Carlo Approach. To avoid the limitations of the pure Monte Carlo approach, we replace the random variables with a low-discrepancy sequence. Because terms in low-discrepancy sequences can be very efficiently enumerated, this approach, up to a small constant, is as fast as the case where the prover can freely provide a sequence of points.

This automatically satisfies the soundness requirement on the output without any additional computational, cryptographic, or statistical assumptions. It provides us with a strong guarantee of the weakest/worst possible bound for the error of the result. Unlike in the failed pure Monte Carlo cases, the prover can search infinitely over sequences and will never find one that breaks soundness. This result can be tight, so additional information about f is required to achieve faster convergence. We briefly mention the potential for improvements with additional information in Section 8.

To avoid leaking information about the computation, we use a Kronecker sequence with public irrational coefficients $\gamma_1, \dots, \gamma_d$ and private seed variables c_1, \dots, c_d ; however, this approach generalizes to any other low-discrepancy sequence which preserves discrepancy when shifted. This provides the prover \mathcal{P} with sufficient freedom to hide the discrete evaluations of $f(x_i)$ from the verifier \mathcal{V} . Up to pathological cases, the verifier \mathcal{V} is prevented from recovering any information about the coefficients c_1, \dots, c_d and any private information about the underlying function f not publicly revealed.

4 Implementation

We implemented our Quasi-Monte Carlo (QMC) zero-knowledge proof framework in `Libsnark` [SCIPR] with a compact domain specific language (DSL) for writing physics and statistical simulations using QMC. This lightweight proof-of-concept zkQMC library on top of `Libsnark` allows for the user to specify a function f , the number of QMC iterations N , the level of precision for calculations, and coefficients $\gamma_1, \dots, \gamma_d$, and it compiles these pieces into a full pipeline for zkSNARK generation and verification. The DSL compiler creates valid `Libsnark` gadgets in C++ which reference smaller hand-optimized gadgets for simple Integer and Fixed-Point arithmetic. This DSL both allows for easier prototyping of zkQMC applications and serves as a proof of concept for automating the conversion of existing QMC programs and codes to a zkQMC framework. A full visual overview of our system architecture is found in Figure 2

In addition to this **lightweight zkQMC library**, we highlight the following three zkQMC applications which we developed for benchmarking and testing: (i) **π estimation**, (ii) **cluster integral approximation**, and (iii) a **simplified particle exposure simulation** using raycasting.

Lightweight zkQMC Library. Our QMC library is divided into two primary components which sit on top of `Libsnark` [SCIPR] which take a function specified in our DSL, f , and number of iterations, N , and produce an arithmetic circuit $C_{qmc,f,N}$ which `Libsnark` uses to generate a zkSNARK, π , attesting to interval, I , s.t.

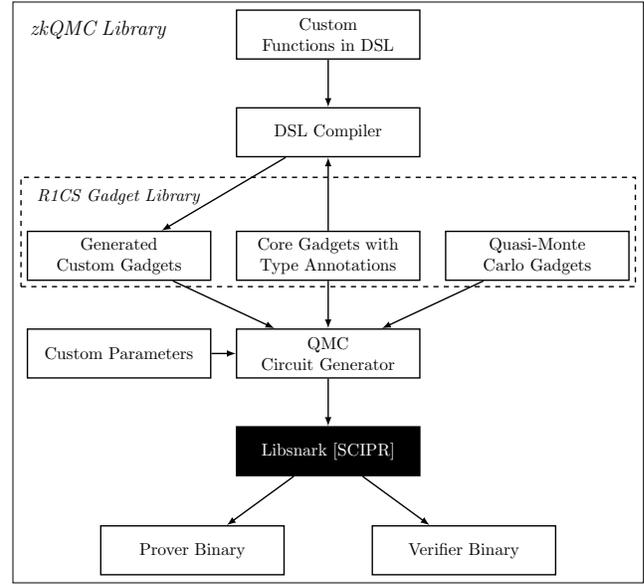


Figure 2: System Architecture Diagram. Everything except the `Libsnark` library is our contribution to the system.

$\int_{[0,1]^d} f(u)du \in I$ and $\lambda_d(I) \in O\left(\frac{(\log N)^d}{N}\right)$. These components are,

- $Compile(f) \rightarrow C_f$, which takes a function specified in our DSL and produces an arithmetic circuit C_f which computes the function using handwritten low-level arithmetic primitives for arithmetic and control flow; and
- $Expand(C_f, N) \rightarrow C_{qmc,f,N}$, which takes the single-iteration arithmetic circuit and produces a final circuit $C_{qmc,f,N}$ that can enter the *Generate*, *Prove*, and *Verify* phases in `Libsnark` or any other NIZK proof system.

The *Compile* phase interprets our DSL and relies on a library of hand-optimized low-level arithmetic primitives with type annotations to transform the readable code into a circuit. This feature was designed so that the introduction of additional primitives immediately provides our DSL language with additional features. The DSL is styled in an imperative form, similar to `BASIC` and `FORTRAN`, and then compiled into an intermediate circuit representation which also includes information on how to compute a satisfying assignment of values for this circuit. The output of this compilation step not just C_f , but also a set of subcircuits of C_f which can be linked and accessed directly by other programs compiled in the same directory.

The *Expand* phase relies on highly templated handwritten glue in C++ to seamlessly connect copies of C_f between constant size lightweight C_{qmc} circuits which transform the i^{th} term in a sequence into the $i+1^{th}$ term

according to the recurrence used to generate the specified low-discrepancy sequence of appropriate dimension. This phase also handles routing variables between iterations, handling elements of the circuit which permit recursive proof composition, and determining private and public variables.

To *expand* \mathcal{C}_f to $\mathcal{C}_{qmc,f,N}$, we introduce $d \times N$ new Field Elements to the proof called $x_{1,1} \cdots x_{d,N}$. We call each tuple $(x_{1,i} \cdots x_{d,i})$ by the shorthand x_i . All elements in x_0 are Unsigned Fixed-Point values which are constrained to the range 0 to 1. All elements in x_{i+1} are generated by copies of a small subcircuit $\mathcal{C}_{qmc}(x_i) \rightarrow x_{i+1}$. The large circuit $\mathcal{C}_{qmc,f,N}$, begins with the constraints on x_0 , then is constructed by interleaving copies of $\mathcal{C}_f(x_i) \rightarrow r_i$ and $\mathcal{C}_{qmc}(x_i) \rightarrow x_{i+1}$ for all i , and finally summing the values of r_i and dividing by N . In the case of recursive proof composition, additional work is required to maintain a counter of the number of iterations between proofs and ensure x_0 is only free when the running count is 0.

Given a program in our DSL, the entire pipeline to generate and verify the zkSNARKs is fully automatic. For benchmarking, we evaluate the time for *Prove* to execute since *Generate* is a one-time setup, *Verify* is cheap, and *Compile* and *Expand* are both one-time and cheap.

The full pipeline is the following:

$$\begin{aligned} \text{Compile}(f) &\rightarrow \mathcal{C}_f \\ \text{Expand}(\mathcal{C}_f, N) &\rightarrow \mathcal{C}_{qmc,f,N} \\ \text{Generate}(\mathcal{C}_{qmc,f,N}) &\rightarrow (pk, vk) \\ \text{Prove}(pk, X, W) &\rightarrow \pi \\ \text{Verify}(vk, (X, \pi)) &\rightarrow \top \vee \perp \end{aligned}$$

Our primary implementation contributions are in the *Compile* and *Expand* functions with `Libsnark` handling the low level proof details.

As with most numeric calculations, there are limitations due to the precision of the point set when this is implemented on a physical computer, but we ignore those limitations for now (they are properly addressed by Niederreiter [Nie92], and ultimately do not change the primary result). For the discussion of the implementation, we ensured that the precision used to store and calculate the points on the prover was sufficient for a large number of trials.

π Estimation. Our first and simplest application of these techniques is one of the classic Monte Carlo demonstrations for estimating the value of π . By choosing quasi-random values in the range $[0, 1]^2$ and counting the proportion which are in the unit circle centered on the origin, we arrive at an approximation of $\frac{\pi}{4}$. In Figure 3 this is the measure of the overlapping area between this circle and quadrant which is shown in black. While both the pseudo-random and quasi-random samples are

expected to approximate $\frac{\pi}{4}$, only quasi-random provides non-trivial worst-case bounds on this constant.

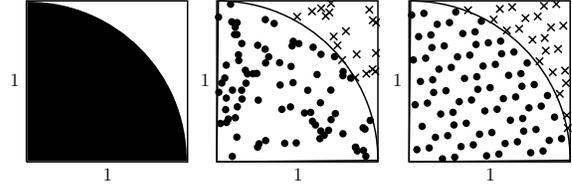


Figure 3: An area of size $\frac{\pi}{4}$ in the unit square (left) and two discrete approximations of this area using Pseudo-Random (middle) and Quasi-Random Points (right).

We can express this in our QMC framework as an area calculated with the following integral:

$$\pi = 4 \int_{[0,1]^2} \mathbb{1}_{\{|v| \leq 1\}} d\mathbf{v}$$

In our DSL, this is a few short lines of code specifying $\mathbb{1}_{\{|v| \leq 1\}}$, the function to be integrated. The zkQMC pipeline handles the task of converting it to R1CS, creating a full QMC circuit, and ultimately producing and verifying a zkSNARK attesting to uncertainty bounds on the integration of the function.

```
# QMC PI iteration using simple gadgets
# Computes x^2 + y^2 <= 1 given (x, y)
FUNC PI_TEST x y -> s
  MUL x x -> x
  MUL y y -> y
  ADD x y -> z
  LEQ z 1 -> s
```

By packing 10,000 QMC iterations into a single proof, we produced a zkSNARK that attests to the following numerical approximation of this integral:

$$4 \int_{[0,1]^2} \mathbb{1}_{\{|v| \leq 1\}} d\mathbf{v} \in [3.093, 3.173].$$

This interval can be made arbitrarily small through additional iterations. While this is a trivial integral which can be analytically computed, it serves as an important example of the technique for demonstration and benchmarking. It is easy to imagine replacing this indicator function with something more complex which potentially also is parameterized by data that the prover wishes to keep secret.

Cluster Integral Approximation. Our next example is one of the first applications of the Monte Carlo technique from “Equations of State Calculations by Fast Computing Machines” [Met+53]. This is the problem of computing cluster integrals, the results of which are used in real gas simulations for handling and approximating

intermolecular interactions. As an integral, problems of this form are expressed in the following way:

$$\int \cdots \int f_{i_1, j_1} \cdots f_{i_{m'}, j_{m'}} \cdots f_{i_m, j_m} d\tau_1 \cdots d\tau_n$$

With the assumption that $f_{i_1, j_1} \cdots f_{i_{m'}, j_{m'}} = 1$, τ_k is the position of particle k , and $f_{i, j}$ is the Mayer f -function (in the hard-sphere model) [May42] [Met+53] on particles i_k and j_k .

$$f_{i_k, j_k} = \begin{cases} 1 & |\tau_{i_k} - \tau_{j_k}| \leq r \\ 0 & |\tau_{i_k} - \tau_{j_k}| > r \end{cases}$$

In this model, we treat the particles as hard-spheres with constant radius which form a bond when they are overlapping. The goal of the full integral is to compute how often a specific configuration of bonds $f_{i_1, j_1} \cdots f_{i_m, j_m}$ occurs given underlying assumptions about a specific subset of this configuration $f_{i_1, j_1}, \dots, f_{i_{m'}, j_{m'}}$. As a visual example, consider the following sample particle configurations on 5 particles in 2 dimensions where the solid lines correspond to assumed bonds and dashed lines correspond to the bonds of interest:

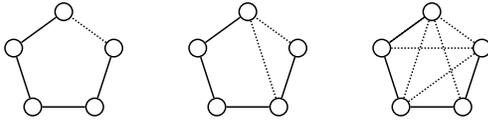


Figure 4: A few sample configurations of 5 particles in 2 dimensions with assumed bonds (bold), and bonds of interest (dashed).

Following the simplifications used in “Equations of State” to reduce the dimensionality of the problem and to transform the domain of the integral from $(-\infty, \infty)^{dn}$ to $[0, 1]^{d(n-2)}$, we implemented this QMC calculation as a series of functions in the DSL where one can customize the Mayer f -function, the dimension, the number of particles, and the bond configuration being checked.

We repeated several of the experiments in “Equations of State Calculations by Fast Computing Machines” [Met+53] in zero knowledge and produced zkSNARKs attesting to the values of various cluster integrals. This application demonstrates the power of this quasi-random technique on more complex integrals. Trivially this only preserves the low-discrepancy sequence in zero knowledge. However, the Mayer f -function can be parameterized by information and assumptions known only to the prover, and in this case that information is also preserved in zero knowledge. It is precisely this sort of parameterization that introduces the non-trivial zero-knowledge guarantees which concern practical applications of this underlying library and technique.

Particle Exposure Simulation. As a larger example, we developed a simplified particle exposure simulation containing a square room, a particle point source (τ), and a hidden room configuration (H) which only the prover knows. In our simulation the room configuration is specified by a list of walls, but the general simulation supports a variety of customizable obstacles by defining additional collision methods in our DSL.

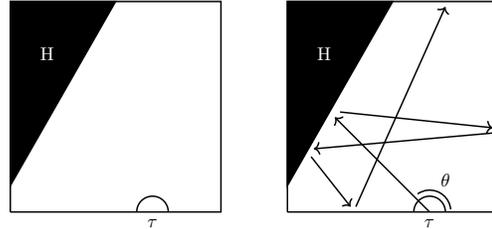


Figure 5: A sample simple physics simulation room configuration (left), and identical room with an example particle trajectory (right).

The prover produces a proof for the verifier that the exposure at various points along the wall opposite to the point source is within certain acceptable bounds by simulating the movement of particles through the room as they bounce of walls and other obstacles. These particles lose energy when they collide with walls and after a fixed number of bounces, they are discarded and their impact on the final result is considered negligible in this simulation.

The room configuration is private to the prover, so the integral which the prover is approximating is also partially hidden from the verifier. While it is not possible for the verifier to directly compute Var_{HK} , it is still possible obtain a reasonable upper-bound on Var_{HK} which can be used to calculate worst-case error bounds. A simplified form of this integral is the following:

$$\int_0^\pi p(\theta, \tau, H) d\theta$$

Where $p(\theta, \tau, H)$ is exposure caused by particles emitted from position τ at angle θ with room configuration H . Note that H and τ are secrets that only the prover knows and θ varies from 0 to π to simulate particles being emitted uniformly in all directions from this source. The prover is forced to attest to a specific pair, (τ, H) , and it is easy to enforce this in zero knowledge.

We developed a complete physics simulation (~ 100 lines of code) in the DSL which calculates trajectories for simulated particles, and our QMC framework uses those gadgets and handles the QMC simulation and communication.

R1CS Encoding and DSL Semantics. In Libsnark, we designed a library of low level operations, called gadgets, which provide instructions on how to generate con-

straints (R1CS) and a satisfying assignment for those constraints for each respective operation implemented. Our DSL is a language agnostic intermediate layer of abstraction to allow for a direct translation from various numeric simulation codes to efficient and correct gadgets that combine our R1CS library functions. In our library of R1CS gadgets, we implement Integer, Fixed-Point, and Boolean types, encoded in Field Elements, and we provide optimized type safe operations on these elements and control flow. As a result of ensuring the correctness of our low level library of gadgets, our compiler is then capable of type-checking the programs written in our DSL, even if they are being translated from an untyped source language.

Additionally, this DSL is still under development, with the goal of correctness and type-safety, so we provide a high level overview of the syntax and semantics, followed by a more detailed discussion of some of our R1CS tricks. Understanding the design of the DSL or the encoding into R1CS is not critical to the main result of paper; however, we provide it here for completeness.

Speaking at a high level, each gadget in this DSL begins with the `FUNC` keyword, followed by its name, and list of variables which are divided into input and output via an arrow `->`. This gadget header is preceded by lines which invoke lower level gadgets which follow a similar syntax. These lower level gadgets include control flow gadgets implemented in our library. All types are inferred at compile-time, as the DSL compiles to a statically-typed intermediate-language in which the primitive operations are defined and typed. We provided a snippet showcasing how succinct writing code in this language can be for a user in our π *Estimation* subsection. Additional details about the specific syntax and semantics of our DSL is provided in Appendix B.

There has been significant work on optimizing the number constraints across a variety of operations in R1CS [Bra12] [Set+12] [KPS18], so we focus here specifically on the implementation of our Fixed-Point operations. Fixed-Point encodings and operations have previously been seen in optimization and machine learning zkSNARK applications; however, the low level implementation of these have not been discussed widely in the literature.

R1CS Encoding of Signed Fixed-Point Numbers. Encoding a Signed Fixed-Point number with h integer bits, and ℓ fractional bits into a Field Element is the same as encoding a Signed Integer in a Field Element with $h + \ell$ bits. However, when generating the circuit, we need to remember the position of the decimal and the ceiling of the number of bits required on either side of it to correctly handle arithmetic and logical operators. The most expensive operations in this model are those that require decomposing the Field Element to its representation in Bits (each individual Field Elements). As a

pragmatic approach to reduce the number of constraints by a small factor, we support allowing the precision to fluctuate, chaining arithmetic operations until just before they risk overflowing.

When adding two numbers with h_1 and h_2 integer bits respectively, the result can be as large as $2^{h_1} + 2^{h_2} - 2$. It is particularly useful here to carefully track the range of possible values since although $2^{h_1} + 2^{h_2} - 2$ requires $\max(h_1, h_2) + 1$ to store, it can still be used across more ADD operations without the number of bits to store the result increasing if h_1 and h_2 are significantly different. When multiplying two numbers with h_1 and h_2 integer bits and ℓ_1 and ℓ_2 fractional bits respectively, the result may require $h_1 + h_2$ integer bits and $\ell_1 + \ell_2$ fractional bits to represent. By dynamically tracking and adjusting precision, instead of reducing to a common word size at the end of each primitive gadget in our library, we reduce the total number of constraints by a constant factor by eliminating and postponing bit decomposition of Field Elements. The precision of more complex operations can be thought about in a similar manner to these basic two, and this optimization complexity is hidden from the programmer by our library and compiler.

R1CS Fixed-Point Number Operation Optimizations. As an example of this fine-grained approach to Fixed-Point subtyping on bits, this representation makes it efficient to compute the next term x_{k+1} of a Kronecker sequence from the current term x_k , where each element of x_k is an Unsigned Fixed-Point number with 0 integer digits and ℓ fractional bits of precision. For each element $x_{i,k}$ of x_k , we compute $\{\gamma_i + x_{i,k}\}$. Provided $0 < \gamma_i < 1$, and γ_i is encoded with ℓ fractional bits of precision, the resulting sum requires ℓ fractional bits and one integer bit to represent, so we form $x_{i,k+1}$ from the bottom ℓ bits of $\gamma_i + x_{i,k}$. This bit tracking and manipulation is handled by our library of gadgets, and the Kronecker sequence constraint generation occurs $N - 1$ times in the $Expand(\mathcal{C}_f, N) \rightarrow \mathcal{C}_{qmc,f,N}$ phase of our pipeline.

As a more complex example, consider the implementations of the `SIN`, and `COS` functions in a gadget—all of which follow the same form, a modulus followed by a Chebyshev polynomial approximation [HWW15]. Our specific implementation depends heavily on the precision required by the calculation, and as this is implemented as an approximation, we must be careful to avoid introducing extra low order fractional bits that could subtly influence the result.

Information about the domain of the function and operations directly prior to the operation allows us to create cleaner and more efficient gadgets. Throughout development, we used these preconditions to enrich our library with extra features and optimizations.

A gadget to enforce the computation of the natural log in the range $[1, 2]$ will require fewer constraints than one for the natural log in the range $[1, 100]$ to achieve

the same accuracy. As a practical examples of this phenomena in action, for some of our implementations, we found that the code called for $\sin(\pi x)$, a MUL followed by the SIN function. Our original SIN implementation began by performing MOD of x by 2π . Knowing that this was immediately preceded by a MUL on x and π , we designed an ISIN gadget which eliminated this operation entirely, replacing it with simpler operations. Specifically, we were able to entirely eliminate the earlier MUL by π and replace the generic MOD of x by 2π at the beginning of SIN with cheaper operations. Instead, at the beginning of the ISIN gadget, we perform a fixed-point division by 2 (which is a free meta-operation from the perspective of constraints since it just involves changing the subtype) and then extract the fractional part of $x/2$.

By adding this to the library, we could immediately refactor our DSL code to include this function. Returning to the Kronecker sequence, combining the modulus in our sequence generation being 1, the preceding addition, and the knowledge that we are working with Unsigned Fixed-Point values in a very specific interval allows us to achieve additional efficiency over the traditional Fixed-Point ADD and MOD gadgets for a similar reason as with the ISIN example. We apply this type of precondition driven optimization through our library for added performance.

5 Benchmarks

We provide benchmarks for our full simulations using quasi-randomness and with the quasi-random constraint removed in Table 1. We evaluate the performance of our implementation by measuring the time and memory required by the prover using 64 threads. Formally we benchmarked proof generation for the full circuit $\mathcal{C}_{qmc,\mathfrak{f},N}$ at various values of N and then a modified form of this circuit $\mathcal{C}_{-,\mathfrak{f},N}$ with the copies of $\mathcal{C}_{qmc}(x_i) \rightarrow x_{i+1}$ removed. In place of each $\mathcal{C}_{qmc}(x_i) \rightarrow x_{i+1}$, we include a $\mathcal{C}_- \rightarrow x_{i+1}$ that allows the prover to non-deterministically sample a value (in an acceptable range). In this comparison we generously assume that any requirements on these random points can be verified outside the circuit to encode the pure Monte Carlo method as efficiently as possible. Even with this relaxation, our $\mathcal{C}_{qmc}(x_i) \rightarrow x_{i+1}$ is only slightly larger than the minimal $\mathcal{C}_- \rightarrow x_{i+1}$ which is reflected by the minor overhead shown in our table.

The focus of our evaluation is on the quantification of this overhead, and the intent of showing these two side-by-side is to demonstrate that the immense benefit to soundness of applying quasi-randomness requires minimal overhead over a fully random unconstrained approach. Additionally, as the simulation gets larger, the cost of adding and enforcing quasi-randomness becomes comparatively smaller, as demonstrated by our results. This is because the overhead introduced by quasi-

randomness in each respective iteration is independent of the total number of iterations. These benchmarks were collected with a circuit using proof-carrying data (PCD) [CT10] which introduces a constant factor overhead on circuit size.

The number of iterations N in Table 1 refers to the number of iterations packed into a single proof. From these numbers, it is possible to extrapolate and estimate the runtime and memory requirements for a given number of iterations packed into a single proof. Alternatively, it is also possible to use these as single prover benchmarks and apply proof-carrying data, maintaining a constant memory requirement in the number of recursive proof generation iterations.

These two approaches have a clear time-memory trade-off. By increasing the number of iterations in each proof, we continue to see a proportionate increase in time and memory resulting from the larger circuit. By increasing the number of proof generation iterations, we see a sharper increase in the amount of time required without the need for any additional memory beyond what it takes to compute a single proof. This sharper increase in time comes from the fixed cost of proof generation and the subcircuit for recursive proof checking.

6 Multi-Prover Systems and Structures

While we have focused primarily on Monte Carlo simulations which can be handled by a single prover with full knowledge, there are also cases where multiple mutually distrustful provers ($\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, etc.) want to prove some result. For this, it is possible to easily extend our techniques to cover many of these cases. In particular, we divide multi-prover setups into two categories which we illustrate with proposed modifications to our base particle exposure simulation. The first is *composition*, where the results from one or more provers running Monte Carlo simulations are used as inputs to another prover running a Monte Carlo simulation. In this scenario each prover has full knowledge of their environment, but the only knowledge outside of it comes as verifiable quantities and intervals from other provers. The second is *distribution*, where several provers want to jointly run a Monte Carlo simulation in a shared environment. The simulation's execution needs to be distributed across these provers because they individually only have pieces of some dataset required to complete the computation.

Composition. For the case of composition, consider a prover \mathcal{P} which wants to calculate $f(u)$ which receives n proofs π_1 through π_n attesting to intervals I_1 through I_d s.t. $u \in I = \prod_{m=1}^d I_m$. Furthermore assume that the function f is sufficiently complex that it cannot be evaluated by standard interval arithmetic. These intervals are not uniform distributions, so the prover cannot treat these as such and use quasi-random se-

Simulation	Iterations	Quasi-Random		Random		Percent QR Overhead	
		Time	Memory	Time	Memory	Time	Memory
<i>π Estimation</i>	10^0	3.0099	0.6892	2.9573	0.6821	1.777%	1.044%
	10^1	3.0946	0.6988	2.9671	0.6920	4.299%	0.988%
	10^2	3.2632	0.7934	3.1136	0.7915	4.802%	0.024%
	10^3	5.6675	1.8916	5.6636	1.8579	0.069%	1.814%
<i>Cluster Integral Approximation</i>	10^0	2.9884	0.7026	2.9775	0.6992	0.037%	0.048%
	10^1	3.1296	0.7528	3.0630	0.7472	2.173%	0.756%
	10^2	4.2717	1.3517	4.2474	1.3353	0.572%	1.226%
	10^3	16.5326	7.1723	15.7250	7.0741	5.136%	1.388%
<i>Particle Exposure Simulation</i>	10^0	3.0686	0.7420	2.9889	0.7368	2.667%	0.709%
	10^1	4.0999	1.2725	4.0704	1.2617	0.725%	0.851%
	10^2	12.2953	6.4352	12.0262	6.4220	2.238%	0.206%
	10^3	127.7580	55.5032	126.4900	55.4283	1.002%	0.135%

Table 1: Libsnark QMC prover performance on our simulations in seconds and gigabytes. The prover is benchmarked on a system with a 2.30GHz Intel Xeon E5-2698 processor on the Darwin cluster [Gar18]. Each experiment was performed 10 times with the table reflecting the average of these executions. Verifier time is approx. 50 ms and is independent of circuit size. Overhead is the percent increase in time/memory going from Random to Quasi-Random. The key observation is how small the QR overhead is in terms of time and memory, especially for more complex computations.

quences to sample from them. However, using a low-discrepancy/quasi-random sequence $x = (x_1, \dots, x_N)$, the prover can achieve the following $O\left(d \frac{\log N}{\sqrt{N}}\right)$ worst-case error bounds on $f(u)$.

$$f(u) \in \left[\min_{x_i \in x} f(x_i) - \Delta, \max_{x_j \in x} f(x_j) + \Delta \right]$$

$$\Delta := d \log(N) \sqrt[4]{B_d \frac{\lambda_d(I)}{N}} \max_{(v_1, \dots, v_n) \in I} \left| \frac{\partial f}{\partial v_1 \dots v_d} \right|.$$

We can easily explicitly find $f(x_i)$, $D_N(x_1, \dots, x_N)$, and $\lambda_d(I)$. An upper bound on $\left| \frac{\partial^d f}{\partial v_1 \dots v_d} \right|$ over I can be found when f is known. The generalization of this approach to cases where f is discontinuous or I is not a hyperbox and the proof of correctness for this algorithm can be found in Appendix C.

Using our particle exposure simulation example, consider the building floor plan in Figure 6. Given that the individual room layouts are secret and particles cannot return to prior rooms, it is natural to ask the question, is the level of exposure safe for room C . Rooms A and B can collaborate to produce a proof of this fact without revealing any information about the layouts of their respective rooms to each other or to C . Room A produces a zkSNARK of a range for the exposure entering room B using our framework. Then, using PCD and the aforementioned technique for using quasi-random sequences to calculate error bounds from uncertainty intervals, room B can use our application to produce a proof that the level of particle exposure in room C is at a safe level. Anyone entering room C can easily verify this proof at any point in time in the future.

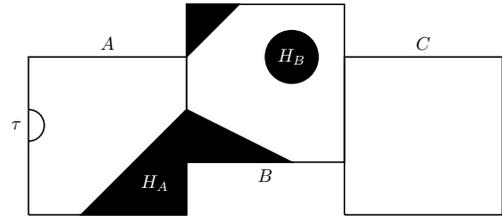


Figure 6: A sample 3-room configuration with a particle emitter in room A for illustrating composition.

Distribution. In the case of distribution, quasi-randomness can be used in the same way as in the single prover case; however, there is a greater concern about breaking zero knowledge between provers which needs to be addressed on a case by case basis. Consider a modified particle exposure example with a single large room where information about the contents of this room is shared across multiple provers such that no individual prover has a full description of the room as in Figure 7.

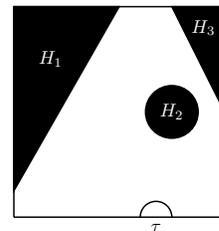


Figure 7: A sample room configuration with 3 obstacles where no prover knows the full layout.

Naïvely computing quasi-random trajectories, check-

ing all provers to find the closest intersection, and applying the QMC physics simulation does not work in this multi-prover distributed case without additional modifications because the information shared between provers can be used to learn about their layout descriptions. This shared information violates the zero knowledge requirement. It is possible to overcome this issue of leaking information on a case by case basis through a restructuring of the underlying algorithm or the introduction of various cryptographic primitives on top of the base QMC setup.

7 Discussion and Conclusion

In this work, we showed how Quasi-Monte Carlo (QMC) techniques help us perform uncertainty quantification in a manner that, unlike with regular Monte Carlo, is publicly verifiable in non-interactive zero knowledge (NIZK) without additional statistical or cryptographic assumptions or machinery beyond what is required for NIZK constructions. We proved and demonstrated its power in distributed non-interactive zero-knowledge settings using recursive proof composition. As shown in our benchmarks and discussion of various quasi-random sequences, the overhead introduced by this QMC technique for each iteration is small in comparison to random alternatives and to the full computation.

QMC is lightweight enough that its addition provides a negligible difference in prover time. However, QMC is powerful enough that even if the prover is able to find a worst-case valid quasi-random sequence of evaluations, the worst-case error bounds will still always contain the true value and these error bounds will be just as tight as if an honest prover used a more desirable quasi-random sequence of the same length.

Using the general lightweight framework we developed, it is possible to choose any quasi-random sequence and any function (with bounded variation) and plug-and-play with the two to immediately begin generating proofs about the integral of that function with worst-case error bounds. We demonstrate this through the implementation of several simple problems in statistics and physics. This framework produces zkSNARKs for numerical integration problems with $O\left(\frac{(\log N)^d}{N}\right)$ precision and can apply our new QMC algorithm for non-uniform interval arithmetic to achieve $O\left(d\frac{\log N}{\sqrt{N}}\right)$ precision with no additional overhead. Assuming the conjecture that $B_d\frac{(\log N)^d}{N}$ is the lowest possible sequence discrepancy [Nie92], then the only way to improve precision with our technique is to introduce additional information about the function being computed.

8 Future Work

Additional work can be done on minimizing the number of QMC iterations to achieve a desired level of error. We believe that producing tighter bounds on the Hardy-Krause variation of a function through modern techniques in symbolic execution or computational measures of Lipschitz continuity may allow for an honest prover to convince a verifier of faster worst-case error convergence. Additionally, reducing the dimension of a problem [IT09] or proving a lower effective dimension as a technique for minimizing QMC problems [WF03] could possibly be incorporated in the prover to improve error bounds. Because the average convergence of QMC is much faster than the worst-case in practice [Laz04] [SC18], it may be possible for an honest prover to be able to convince a verifier of better error bounds by using additional undiscovered or unexplored techniques.

Through these advances, it may be possible to generalize these techniques beyond the Monte Carlo method to the inexpensive zero-knowledge verification of all randomized polynomial time algorithms with minimal additional statistical assumptions [Cha00], including those where Var_{HK} is exponential in d , without full derandomization.

9 Acknowledgements

This paper was improved by discussions with and revisions from the following people for whose feedback the authors are deeply grateful: Katherine Casamento, Juston Moore, David Mordecai, Michael Walfish, and many others. This research was supported by the Information Science and Technology Institute (ISTI) of Los Alamos National Laboratory (LANL) under project number 20210529CR and by LANL’s Nuclear Weapons Cyber Assurance Laboratory (NWCAL). LANL is operated by Triad National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy (Contract No. 89233218CNA000001). Approved for unlimited public release: LA-UR-22-28108.

References

- [Adl78] Leonard Adleman. “Two theorems on random polynomial time”. In: *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*. 1978, pp. 75–83.
- [Ais+16] Christoph Aistleitner et al. *On functions of bounded variation*. 2016.
- [Ang+21] Sebastian Angel et al. *Efficient Representation of Numerical Optimization Problems for SNARKs*. Cryptology ePrint Archive, Report 2021/1436. 2021.
- [Bab79] László Babai. *Monte-Carlo algorithms in graph isomorphism testing*. 1979.

- [BBG97] Phelim Boyle, Mark Broadie, and Paul Glasserman. “Monte Carlo methods for security pricing”. In: *Journal of Economic Dynamics and Control* 21.8 (1997), pp. 1267–1321.
- [Bec94] Jozsef Beck. “Probabilistic Diophantine Approximation, I. Kronecker Sequences”. In: *Annals of Mathematics* 140.2 (1994), pp. 449–502.
- [Ben+13] Eli Ben-Sasson et al. *SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge*. Cryptology ePrint Archive, Report 2013/507. 2013.
- [Ben+14a] Eli Ben-Sasson et al. *Scalable Zero Knowledge via Cycles of Elliptic Curves*. Cryptology ePrint Archive, Report 2014/595. 2014.
- [Ben+14b] Eli Ben-Sasson et al. “Zerocash: Decentralized Anonymous Payments from Bitcoin”. In: *IACR Cryptology ePrint Archive* 2014 (2014), p. 349.
- [BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. “Non-Interactive Zero-Knowledge and Its Applications”. In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC ’88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 103–112.
- [BG81] Charles H. Bennett and John Gill. “Relative to a Random Oracle A, PA != NPA != co-NPA with Probability 1”. In: *SIAM J. Comput.* 10 (1981), pp. 96–113.
- [BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Report 2019/1021. 2019.
- [BH10] Kurt Binder and Dieter W. Heermann. *Monte Carlo Simulation in Statistical Physics*. Graduate Texts in Physics. Springer, 2010.
- [Bit+11] Nir Bitansky et al. *From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again*. Cryptology ePrint Archive, Report 2011/443. 2011.
- [Bra12] Benjamin Braun. “Compiling computations to constraints for verified computation”. In: 2012.
- [Caf98] Russel E. Caflisch. “Monte Carlo and quasi-Monte Carlo methods”. In: *Acta Numerica* 7 (1998), pp. 1–49.
- [Cha00] Bernard Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, 2000.
- [CHL15] Mathieu Cambou, Marius Hofert, and Christiane Lemieux. *Quasi-random numbers for copula models*. 2015.
- [CT10] Alessandro Chiesa and Eran Tromer. “Proof-Carrying Data and Hearsay Arguments from Signature Cards.” In: *Innovations in Computer Science Proceedings*. Jan. 2010, pp. 310–331.
- [CTV15] Alessandro Chiesa, Eran Tromer, and Madars Virza. *Cluster Computing in Zero Knowledge*. Cryptology ePrint Archive, Report 2015/377. 2015.
- [Cun+14] Americo Cunha et al. “Uncertainty quantification through the Monte Carlo method in a cloud computing setting”. In: *Computer Physics Communications* 185.5 (2014), pp. 1355–1363.
- [DEM96] David P. Dobkin, David Eppstein, and Don P. Mitchell. “Computing the Discrepancy with Applications to Supersampling Patterns”. In: *ACM Trans. Graph.* 15.4 (1996), pp. 354–376.
- [DeS20] Zachary DeStefano. *Privacy Preserving, Distributed, and Verifiable Machine Learning for COVID-19 Identification using Zero-Knowledge Proofs*. Chesapeake Large-Scale Analytics Conference. 2020.
- [DGW14] Carola Doerr, Michael Gnewuch, and Magnus Wahlström. “Calculation of Discrepancy Measures and Applications”. In: *Lecture Notes in Mathematics* (2014), pp. 621–678.
- [Gar18] Charles Kristopher Garrett. “The Darwin Cluster”. In: (2018).
- [GGPR] Rosario Gennaro et al. *Quadratic Span Programs and Succinct NIZKs without PCPs*. Cryptology ePrint Archive, Report 2012/215. 2012.
- [Gil77] John Gill. “Computational Complexity of Probabilistic Turing Machines”. In: *SIAM Journal on Computing* 6.4 (1977), pp. 675–695.
- [GMR85] S Goldwasser, S Micali, and C Rackoff. “The Knowledge Complexity of Interactive Proof-Systems”. In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC ’85. Providence, Rhode Island, USA: Association for Computing Machinery, 1985, pp. 291–304.
- [GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. “Proofs That Yield Nothing but Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems”. In: *J. ACM* 38.3 (July 1991), pp. 690–728.
- [Groth10] Jens Groth. “Short Pairing-Based Non-interactive Zero-Knowledge Arguments”. In: *Advances in Cryptology - ASIACRYPT 2010*. Ed. by Masayuki Abe. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 321–340.
- [Groth16] Jens Groth. *On the Size of Pairing-based Non-interactive Arguments*. Cryptology ePrint Archive, Report 2016/260. 2016.
- [GSW09] Michael Gnewuch, Anand Srivastav, and Carola Winzen. “Finding optimal volume subintervals with k points and calculating the star discrepancy are NP-hard problems”. In: *Journal of Complexity* 25.2 (2009), pp. 115–127.

- [HK21] Julien Hok and Sergei Kucherenko. *Pricing and Risk Analysis in Hyperbolic Local Volatility Model with Quasi Monte Carlo*. 2021.
- [Hla61] E. Hlawka. “Funktionen von beschränkter Variatiou in der Theorie der Gleichverteilung”. In: *Annali di Matematica Pura ed Applicata* 54 (1961), pp. 325–333.
- [HWW15] Cecil Hastings, Jeanne T. Wayward, and James P. Wong. *Approximations for Digital Computers*. Princeton University Press, 2015.
- [IT09] Junichi Imai and Ken Seng Tan. “Dimension reduction approach to simulating exotic options in a Meixner Levy market”. In: *IAENG International Journal of Applied Mathematics* 39 (Nov. 2009), pp. 1–11.
- [Kil92] Joe Kilian. “A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)”. In: *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*. STOC ’92. Victoria, British Columbia, Canada: Association for Computing Machinery, 1992, pp. 723–732.
- [Ko+20] Hankyung Ko et al. *Efficient Verifiable Image Redacting based on zk-SNARKs*. Cryptology ePrint Archive, Report 2020/1579. 2020.
- [KPS18] Ahmed Kosba, Charalampos Papamanthou, and Elaine Shi. “xJsnark: A Framework for Efficient Verifiable Computation”. In: *2018 IEEE Symposium on Security and Privacy*. May 2018, pp. 944–961.
- [Kro+14] Dirk P. Kroese et al. “Why the Monte Carlo method is so important today”. In: *WIREs Computational Statistics* 6.6 (2014), pp. 386–392.
- [KW97] Ladislav Kocis and William J. Whiten. “Computational Investigations of Low-Discrepancy Sequences”. In: 23.2 (July 1997), pp. 266–294.
- [Laz04] Achilleas Lazopoulos. “Error estimates in Monte Carlo and Quasi-Monte Carlo integration”. In: *Acta Physica Polonica B* 35 (Nov. 2004).
- [Lee+20] Seunghwa Lee et al. *vCNN: Verifiable Convolutional Neural Network based on zk-SNARKs*. Cryptology ePrint Archive, Report 2020/584. <https://ia.cr/2020/584>. 2020.
- [LL03] G Larcher and G Leobacher. “Quasi-Monte Carlo and Monte Carlo methods and their application in finance”. In: *Surveys on Mathematics for Industry* 11 (Jan. 2003).
- [LN93] Gerhard Larcher and Harald Niederreiter. “Kronecker-type sequences and nonarchimedean diophantine approximations”. eng. In: *Acta Arithmetica* 63.4 (1993), pp. 379–396.
- [Mal+19] Mary Maller et al. *Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings*. Cryptology ePrint Archive, Report 2019/099. 2019.
- [May42] Joseph E. Mayer. “Contribution to Statistical Mechanics”. In: *The Journal of Chemical Physics* 10.10 (1942), pp. 629–643.
- [Met+53] N. Metropolis et al. “Equation of state calculations by fast computing machines”. In: *Journal of Chemical Physics* 21 (1953), pp. 1087–1092.
- [Mic00] Silvio Micali. “Computationally Sound Proofs”. In: *SIAM J. Comput.* 30.4 (2000), pp. 1253–1298.
- [MU49] Nicholas Metropolis and S. Ulam. “The Monte Carlo Method”. In: *Journal of the American Statistical Association* 44.247 (1949). PMID: 18139350, pp. 335–341.
- [Nie92] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Philadelphia: Society for Industrial and Applied Mathematics, 1992.
- [Owe03] Art B. Owen. “Quasi-Monte Carlo Sampling”. In: *SIGGRAPH 2003*. 2003.
- [Owe04] A. Owen. “Multidimensional variation for quasi-Monte Carlo”. In: 2004.
- [Pap03] A. Papageorgiou. “Sufficient conditions for fast quasi-Monte Carlo convergence”. In: *Journal of Complexity* 19.3 (2003). Oberwolfach Special Issue, pp. 332–351.
- [Par+13] Bryan Parno et al. “Pinocchio: Nearly Practical Verifiable Computation”. In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 238–252.
- [Ram81] Lyle Ramshaw. “On the discrepancy of the sequence formed by the multiples of an irrational number”. In: *Journal of Number Theory* 13.2 (1981), pp. 138–175.
- [Ric51] R D Richtmyer. “The Evaluation Of Definite Integrals, And A Quasi-Monte-Carlo Method Based On The Properties Of Algebraic Numbers”. In: (Oct. 1951).
- [SC18] Tobias Schwedes and Ben Calderhead. *Quasi Markov Chain Monte Carlo Methods*. 2018.
- [SCIPR] SCIPR Lab. *libsark: a C++ library for zkSNARK proofs*. <https://github.com/scipr-lab/libsark>.
- [SEB08] Peter Shirley, Dave Edwards, and Solomon Boulos. “Monte Carlo and Quasi-Monte Carlo Methods for Computer Graphics”. In: *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Ed. by Alexander Keller, Stefan Heinrich, and Harald Niederreiter. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 167–177.
- [Set+12] Srinath Setty et al. “Taking Proof-Based Verified Computation a Few Steps Closer to Practicality”. In: *21st USENIX Security Symposium (USENIX Security 12)*. Bellevue, WA: USENIX Association, Aug. 2012, pp. 253–268.

- [Set20] Srinath Setty. “Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup”. In: *Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III*. Santa Barbara, CA, USA: Springer-Verlag, 2020, pp. 704–737.
- [Tuf04] Bruno Tuffin. “Randomization of Quasi-Monte Carlo Methods for Error Estimation: Survey and Normal Approximation*”. In: 10.3-4 (2004), pp. 617–628.
- [Wah+17] Riad S. Wahby et al. *Doubly-efficient zk-SNARKs without trusted setup*. Cryptology ePrint Archive, Report 2017/1132. 2017.
- [Wei00] Stefan Weinzierl. *Introduction to Monte Carlo methods*. 2000.
- [Wey16] Hermann Weyl. “Über die Gleichverteilung von Zahlen mod. Eins”. In: *Mathematische Annalen* 77.3 (1916), pp. 313–352.
- [WF03] Xiaoqun Wang and Kai-Tai Fang. “The Effective Dimension and Quasi-Monte Carlo integration”. In: *Journal of Complexity* 19 (Apr. 2003), pp. 101–124.
- [Zha21] Jiaxin Zhang. “Modern Monte Carlo methods for efficient uncertainty quantification and propagation: A survey”. In: *WIREs Computational Statistics* 13.5 (2021), e1539.

A BPP, RP, ZPP, and Las Vegas Algorithms in Zero Knowledge

Previous sections primarily focus on techniques for capturing the results of Monte Carlo methods on functional problems with zero-knowledge proofs; however, this machinery can be slightly modified for attesting to the outcomes of decision problems. We developed the following general transformations for embedding programs which decide languages in various probabilistic complexity classes in interactive and non-interactive zero-knowledge proofs. Only some of the classes we describe require QMC. For completeness, we discuss the application of QMC to subsets of BPP, RP, and co-RP. Additionally we discuss faster general techniques for succinctly embedding Las Vegas algorithms in ZKPs which is efficient in expectation for the prover.

QMC For BPP. BPP (Bounded-Error Probabilistic Polynomial-Time) is the class of all decision problems that can be solved in polynomial time with randomness with a two-sided error of $1/3$ [Gil77]. This is to say, there exists an algorithm \mathcal{A} which for a particular input x and a random input \mathbf{r} , accepts with probability $2/3$ if $x \in L$ and accepts with probability no more than $1/3$ if $x \notin L$.

By a generalization of Adleman’s Theorem [Adl78], there exists a derandomized family of polynomial-sized

circuits which can decide the membership of $x \in L$ for any $L \in \text{BPP}$ [BG81], but there is no known efficient procedure for producing these derandomized circuits. We describe an efficient procedure using QMC for partially derandomizing a specific subset of decision problems in BPP so membership can be proved in zero knowledge. In a way, this QMC solution combines the best qualities of the fully derandomized circuits with the original probabilistic algorithm.

Both the fully derandomized circuit and the QMC circuit, have perfect soundness prior to being embedded in a ZKP; however, unlike a QMC equipped circuit, a fully derandomized circuit is often prohibitively expensive to produce. Both the probabilistic algorithm and the QMC circuit are polynomial time to compute, but unlike the probabilistic algorithm, the QMC circuit has perfect soundness. For the subset of BPP where QMC can be efficiently applied and no fully derandomized polynomial-sized circuits are known, we demonstrate a general procedure for embedding these randomized computations in ZKPs.

As previously discussed, QMC is a technique for taking an integral and producing worst case error bounds on the true value of that integral. To apply QMC to a decision problem in BPP, we need to rephrase our decision problem as integral. By a slight abuse of notation, we say that \mathcal{A}_x is the function which, defined on a particular input x , and using d random values \mathbf{r} , outputs either 0 or 1 which correspond to a verdict of exclusion or inclusion respectively of x in L . We can then use QMC to compute bounds \mathcal{I} on the following integral:

$$\int_{[0,1]^d} \mathcal{A}_x(\mathbf{r}) d\mathbf{r} = \begin{cases} \geq 2/3 & x \in L \\ \leq 1/3 & x \notin L \end{cases}.$$

To turn our QMC approximation of the integral into a decision, we need to iterate until the uncertainty bound, \mathcal{I} , is small enough to completely exclude one of the ranges $[0, 1/3]$ or $[2/3, 1]$. If $\mathcal{I} \cap [0, 1/3]$ is empty then accept, and if $\mathcal{I} \cap [2/3, 1]$ is empty then reject.

Recall that this integral can only be efficiently estimated with QMC techniques when $\text{Var}_{HK}(\mathcal{A}_x)$ is sufficiently small, limiting us to a specific subset of BPP. Specifically, we require that

$$\text{Var}_{HK}(\mathcal{A}_x) < \frac{N}{6B_d(\log N)^d}$$

for a predetermined d and small constant B_d to decide membership of x in L in N iterations.

This follows directly from the QMC worst case error bound applied to \mathcal{A}_x . Specifically we know that

$$|\mathcal{I}| := 2B_d \frac{(\log N)^d}{N} \text{Var}_{HK}(\mathcal{A}_x),$$

and in the worst case it is required that $|\mathcal{I}| < 1/3$ to arrive at a decision. This gives use the following inequality

$$2B_d \frac{(\log N)^d}{N} \text{Var}_{HK}(\mathcal{A}_x) < 1/3$$

which can be trivially rearranged to appear as a condition on $\text{Var}_{HK}(\mathcal{A}_x)$.

QMC For RP and co-RP. We can adapt the above approach for two-sided error problems to one-sided probabilistic decision problems in RP and co-RP. Without loss of generality, we limit the discussion below to RP as the argument for co-RP is similar. RP is the class of all decision problems that can be solved in polynomial time with randomness which always reports $x \notin L$ correctly, but errors with probability $\leq 1/2$ when $x \in L$ [Gil77]. Given an algorithm \mathcal{A}_x defined similarly as in the BPP case, we can express RP similarly as the integral

$$\int_{[0,1]^d} \mathcal{A}_x(\mathbf{r}) d\mathbf{r} = \begin{cases} \geq 1/2 & x \in L \\ 0 & x \notin L \end{cases}.$$

When we embed this integral in a zero-knowledge proof, because the error is only one-sided, we only need to apply QMC when $\mathcal{A}_x(r)$ exclusively reports 0 (the RP algorithm rejects). That is, we need to run enough QMC iterations to reduce the interval bound, \mathcal{I} , to exclude $[1/2, 1]$. If revealing the decision does not violate zero knowledge, and we can find a witness that causes the RP algorithm to accept, we can simply prove that such a witness exists without resorting to QMC techniques.

In the case of RP and co-RP, the QMC approach is only efficient when $\text{Var}_{HK}(\mathcal{A}_x)$ is sufficiently small:

$$\text{Var}_{HK}(\mathcal{A}_x) < \frac{N}{2B_d(\log N)^d}.$$

Because the error is one-sided, $\text{Var}_{HK}(\mathcal{A}_x)$ can be twice as large.

Las Vegas Algorithms in Zero Knowledge. We showed how QMC techniques can be applied to some decision problems with probabilistic polynomial time algorithms and one and two-sided error; however, during our exploration of embedding randomized computations in NIZKs, we also discovered an easy and efficient method for handling zero-sided error randomized computations, namely Las Vegas algorithms [Bab79] (which include all decision problems in ZPP).

As opposed to Monte-Carlo algorithms which run in polynomial time and produce a result with a certain quantifiable degree of uncertainty, Las Vegas algorithms *always* produce a correct result, but vary in runtime that is polynomial in expectation (over different choices in random inputs).

A computation with unbounded or variable length cannot be naively encoded in constraints using standard techniques like unrolling since zero-knowledge proof frameworks typically require fixed sized constraint systems. We overcome this issue by using a short encoding of the algorithm in constraints to certify executions without resorting to program unrolling. Additionally, we

ensure that the prover is expected to successfully find a valid witness in the original expected runtime of the algorithm multiplied by some small constant.

We encode, in constraints, a trace through the algorithm that must terminate in fewer than $c \cdot \mathbb{E}[t_X(\mathbf{R})]$ steps where c is a predetermined constant and $\mathbb{E}[t_X(\mathbf{R})]$ is the expected length of a trace over all inputs X and random values \mathbf{R} . The first job of the prover is to iterate over random values (potentially in parallel) to find a *random* input \mathbf{r} which produces a trace of length less than $c \cdot \mathbb{E}[t_X(\mathbf{R})]$. If the prover reaches the $c \cdot \mathbb{E}[t_X(\mathbf{R})]$ th step of any trace without halting, that random value, \mathbf{r}_{i-1} , can be discarded and a new one, \mathbf{r}_i , tried until a successful one is found. As a note, if we intuitively assume that $\mathbb{E}[t_x(\mathbf{R})] \approx \mathbb{E}[t_X(\mathbf{R})]$ for any $x \in X$, all of the following probabilities and expectations effectively only range over the choice of random values (not over the the inputs X). Also, the value of c can be tuned to balance the length of the trace being encoded in constraints with the expected number of times the prover needs to choose a new random value to test.

The expected number N_c of independent random values, modeled by the independent random variables $\{\mathbf{R}_i\}_{i=1}^{\infty}$, that the prover will need to search through to find a trace of length less than $c \cdot \mathbb{E}[t_X(\mathbf{R})]$ is

$$N_c = \frac{1}{1 - \mathbb{P}[t_X(\mathbf{R}_i) \geq c \cdot \mathbb{E}[t_X(\mathbf{R})]]}$$

by a well known result on the expected value of geometric random variables.

Because $t_x(\mathbf{r}_i) > 0$ for all \mathbf{r}_i , we can apply Markov's Inequality, which says,

$$\mathbb{P}[t_X(\mathbf{R}_i) \geq c \cdot \mathbb{E}[t_X(\mathbf{R})]] \leq \frac{\mathbb{E}[t_X(\mathbf{R}_i)]}{c \cdot \mathbb{E}[t_X(\mathbf{R})]} = \frac{1}{c}.$$

This gives us the following after simplification

$$N_c \leq \frac{c}{c-1}$$

which is defined for all $c > 1$.

Intuitively, smaller values of c require the prover to search through more traces in expectation, but they require fewer constraints to encode. For this proof we can arbitrarily choose $c = 2$ and evaluate for this specific value of c to get $N_2 \leq 2$. In expectation the prover will only need to search through at most two traces to find one with length smaller than $2 \cdot \mathbb{E}[t_X(\mathbf{R})]$.

The value of c can be chosen judiciously by the prover to minimize the total amount of computation required. Often producing a particular trace is significantly less computationally expensive than encoding that same length trace in a proof. Additionally, we note that for particular Las Vegas algorithms, it may be more efficient to constrain the verification of the result and not the full algorithm's execution, but this serves as a more general procedure for cases where that is not possible.

B DSL Syntax and Semantics

Our domain specific language, used for constructing higher level gadgets in our zkQMC library, is described by the grammar in Figure 8 and the operational semantics described in Figure 9. In our language, a gadget is defined by a *function definition*, which consists of the word “FUNC”, followed by the name of the function, a list of parameter names, a “->” symbol, and the variables to be returned. This is then followed by *Assignments*, consisting of either a function name and a list of inputs, or a constant, followed by a “->” which binds the value to the variable to the right of the arrow.

$Def ::=$	function definition
$\text{FUNC } x \ x^* \rightarrow o^* \ \backslash n\{a\backslash n\}^*$	
$a ::=$	assignment
$\{\{ x \ i^* \} \mid n \} \rightarrow o^*$	
$i ::=$	input
x	
n	
$o ::=$	output variable
x	
$v ::=$	value
n	
$n ::=$	number
$-?\{[0..9]\} + \{.\{[0..9]\}+\}?$	
$x ::=$	variable
$\{[A..Z][a..z] _ \}+$	

Figure 8: DSL Grammar

These assignments must only reference variables introduced in the parameter list of the function, or variables introduced in earlier assignments. This language supports shadowing, where additional variables with the same name can be introduced, with subsequent references referring to the most recent assignment.

$$\begin{array}{c}
 \frac{}{\langle n, \sigma \rangle \Downarrow n} \text{LIT} \quad \frac{\sigma(x) = v}{\langle x, \sigma \rangle \Downarrow v} \text{VAR} \quad \frac{}{\langle [], \sigma \rangle \Downarrow []} \text{NIL} \\
 \\
 \frac{\langle i, \sigma \rangle \Downarrow v_i \quad \langle is, \sigma \rangle \Downarrow v_{is}}{\langle i :: is, \sigma \rangle \Downarrow v_i :: v_{is}} \text{CONS} \\
 \\
 \frac{\sigma(f) = \text{FUNC } f [x_1, \dots, x_n] \rightarrow [o_1, \dots, o_m] \ a \quad \langle a, \sigma[x_1 \mapsto v_1, \dots, x_n \mapsto v_n] \rangle \Downarrow \sigma' \quad [\sigma'(o_1), \dots, \sigma'(o_m)] = [v_{o1}, \dots, v_{om}]}{\langle f [v_1, \dots, v_n], \sigma \rangle \Downarrow [v_{o1}, \dots, v_{om}]} \text{CALL} \\
 \\
 \frac{\langle [i_1, \dots, i_n], \sigma \rangle \Downarrow [v_1, \dots, v_n] \quad \langle f [v_1, \dots, v_n], \sigma \rangle \Downarrow [v_{o1}, \dots, v_{om}] \quad \langle a, \sigma[o_1 \mapsto v_{o1}, \dots, o_m \mapsto v_{om}] \rangle \Downarrow \sigma'}{\langle f [i_1, \dots, i_n] \rightarrow [o_1, \dots, o_m] :: a, \sigma \rangle \Downarrow \sigma'} \text{ASSIGN CONS} \\
 \\
 \frac{}{\langle [], \sigma \rangle \Downarrow \sigma} \text{ASSIGN NIL}
 \end{array}$$

Figure 9: DSL Operational Semantics

These variables and assignments are evaluated according to the following rules. The arguments are bound to the parameter names in σ . Each assignment evaluates the left-hand sign of the “->”, and adds a binding from that to the variable on the right to the context σ . Once all of the assignments have been loaded into the context, the return variable is looked up in the context, and that value is returned.

Functions of this form can be translated into **Libsnark** gadgets via the prototype DSL compiler. This compiler converts DSL variables to **Libsnark** variables by first substituting the variable to which the constant is bound for the associated literal through the rest of the assignments, and then allocating the remainder as protoboard variables. The functions are then converted to gadgets, where the gadget is parameterized by the matching variables, with the last variable as the “output” variable.

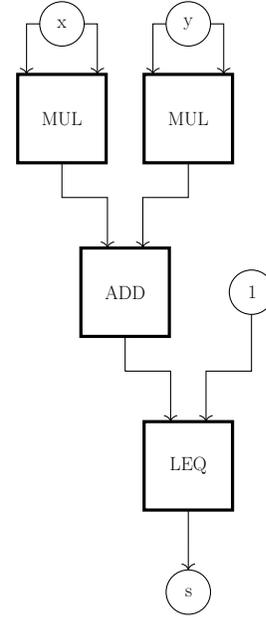


Figure 10: A Representation of PI_TEST Gadget in a High-Level Circuit Form.

This results in a circuit implemented in **Libsnark** matching the function defined in the DSL. Recall that these gadgets are responsible for both generating RICS constraints and a satisfying assignment for these constraints. As an example, consider running this compiler on the π estimation program defined in Section 4. This results in a **Libsnark** gadget called PI_TEST represented by Figure 10. Gadgets created by compilation using our DSL and compiler can call any **Libsnark** gadgets as functions, both those created from compilation from the DSL and those which are implemented manually. For this larger zkQMC library, an initial library of gadgets was manually written, optimized, and then linked together in our DSL to create larger gadgets and ultimately full zkQMC applications.

C Interval Algorithm Correctness Proof

Theorem 1. *Let I be a closed d -dimensional hyperbox and $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a function which is Lipschitz continuous over I , then*

$$u \in I \implies f(u) \in \left[\min_{x_i \in x} f(x_i) - \Delta, \max_{x_j \in x} f(x_j) + \Delta \right]$$

where

$$\Delta := \frac{1}{2} d \sqrt[d]{D_N(x_1, \dots, x_N) \lambda_d(I)} \max_{(v_1, \dots, v_d) \in I} \left| \frac{\partial f}{\partial v_1 \dots v_d} \right|$$

for any arbitrary set of N points $X = \{x_1, \dots, x_N\} \subset I$

Proof. As motivation for the following proof, $\lambda_d(I)$ is a scaling factor to transform $[0, 1]^d$ into I , and the true minimum and maximum in this scaled down interval are located no further than $\frac{1}{2} D_N(x_1, \dots, x_N)$ away (L_1 distance) from some x_i and x_j respectively, so a limit on the local derivative of the function over the interval corresponds directly to a limit on the minimum and maximum over the interval.

By definition, we have

$$f(u) \in \left[\min_{v \in I} f(v), \max_{v \in I} f(v) \right].$$

By definition of Lipschitz continuity over I , we have

$$|f(x) - f(v')| \leq K |x - v'|$$

$$K := \max_{(v_1, \dots, v_n) \in I} \left| \frac{\partial^d f}{\partial v_1 \dots v_d} \right|$$

Let $v' \in I$ be an arbitrary coordinate (v'_1, \dots, v'_d) s.t.

$$f(v') = \min_{v \in I} f(v) \text{ or } f(v') = \max_{v \in I} f(v).$$

Let I' be a hyperbox centered on v' with side length

$$\sqrt[d]{D_N(x_1, \dots, x_N) \lambda_d(I)} + \epsilon \text{ where } \epsilon > 0.$$

Suppose, by contradiction, that this box does not contain any points in X . Additionally, let \mathcal{I} be the set of all subintervals of I .

$$D_N(x_1, \dots, x_N) = \sup_{B \in \mathcal{J}} \left| \frac{A(B; X)}{N} - \lambda_d(B) \right|$$

$$D_N(x_1, \dots, x_N) \lambda_d(I) = \sup_{B \in \mathcal{I}} \left| \frac{A(B; X)}{N} - \lambda_d(B) \right|$$

$$D_N(x_1, \dots, x_N) \lambda_d(I) \geq \left| \frac{0}{N} - \lambda_d(I') \right|$$

$$D_N(x_1, \dots, x_N) \lambda_d(I) \geq \left(\sqrt[d]{D_N(x_1, \dots, x_N) \lambda_d(I)} + \epsilon \right)^d$$

$$D_N(x_1, \dots, x_N) \lambda_d(I) \geq D_N(x_1, \dots, x_N) \lambda_d(I) + \epsilon$$

$$0 \geq \epsilon$$

We reach a contradiction $0 \geq \epsilon$, therefore there must exist a $x' \in I' \cap X$, and furthermore,

$$|x' - v'| \leq \frac{1}{2} d \sqrt[d]{D_N(x_1, \dots, x_N) \lambda_d(I)}$$

by placing x' at the corner of the hyperbox.

To complete the proof we reintroduce our Lipschitz continuity condition,

$$|f(x') - f(v')| \leq K |x' - v'|$$

$$\leq \frac{1}{2} K d \sqrt[d]{D_N(x_1, \dots, x_N) \lambda_d(I)}.$$

If v' is a minimum, then we have $f(x) \geq f(v')$, so

$$f(v') \geq f(x') - \frac{1}{2} K d \sqrt[d]{D_N(x_1, \dots, x_N) \lambda_d(I)}.$$

If v' is a maximum, then we have $f(x) \leq f(v')$, so

$$f(v') \leq f(x') + \frac{1}{2} K d \sqrt[d]{D_N(x_1, \dots, x_N) \lambda_d(I)}.$$

The expanded form just substitutes $\left| \frac{\partial f}{\partial v_1 \dots v_d} \right|$ for K \square

As a consequence of Theorem 1 and by using a low-discrepancy sequence

$$D_N^*(x_1, \dots, x_n) = B_d \frac{(\log N)^d}{N}$$

we have the following corollary.

Corollary 1. *With suitable choice of X*

$$\Delta \in O\left(d \frac{\log N}{\sqrt[d]{N}}\right),$$

$$\Delta := d \log(N) \sqrt[d]{B_d \frac{\lambda_d(I)}{N}} \max_{(v_1, \dots, v_n) \in I} \left| \frac{\partial f}{\partial v_1 \dots v_d} \right|.$$

We can generalize the above corollary on hyperboxes I to include arbitrary convex regions \mathcal{R} by swapping the mapping from $I \rightarrow [0, 1]^d$ for a more general transformation function $f_t : \mathcal{R} \rightarrow [0, 1]^d$. Formally the scaling factor $\lambda_d(I)$ becomes

$$\max_{v \in \mathcal{R}} \left| \det \left(\mathbf{J}_{f_t^{-1}}(v) \right) \right|$$

which we shorthand as $\Lambda_{\mathcal{R}, f_t}$. As a consequence of this and our discussion of segmented QMC, we have the following result.

Corollary 2. *If f in the region \mathcal{R} can be decomposed into finitely many (s) Lipschitz continuous convex regions I_1, \dots, I_s , the set of which we call \mathcal{S} we have*

$$\Delta := \max_{I_i \in \mathcal{S}} (\Delta_i),$$

$$\Delta_i := d \log(N) \sqrt[d]{B_d \frac{\Lambda_{I_i, f_t}}{N}} \max_{(v_1, \dots, v_n) \in I_i} \left| \frac{\partial f}{\partial v_1 \dots v_d} \right|.$$