Zero-Knowledge Location Privacy via Accurate Floating-Point SNARKs

Jens Ernstberger*§, Chengru Zhang†§, Luca Ciprian*, Philipp Jovanovic‡, and Sebastian Steinhorst*
*Technical University of Munich, Germany
†The University of Hong Kong, Hong Kong
‡University College London, United Kingdom

***Abstract*—We introduce Zero-Knowledge Location Privacy (ZKLP), enabling users to prove to third parties that they are within a specified geographical region while not disclosing their exact location. ZKLP supports varying levels of granularity, allowing for customization depending on the use case. To realize ZKLP, we introduce the first set of Zero-Knowledge Proof (ZKP) circuits that are fully compliant to the IEEE 754 standard for floating-point arithmetic.**

Our results demonstrate that our floating point circuits amortize efficiently, requiring only 64 constraints per operation for 2^{15} single-precision floating-point multiplications. We utilize our floating point implementation to realize the ZKLP paradigm. In comparison to a baseline, we find that our optimized implementation has $15.9\times$ less constraints utilizing single precision floating-point values, and $12.2\times$ less constraints when utilizing double precision floating-point values. We demonstrate the practicability of ZKLP by building a protocol for *privacy preserving peer-to-peer proximity testing* — Alice can test if she is close to Bob by receiving a single message, without either party revealing any other information about their location. In such a setting, Bob can create a proof of (non-)proximity in $0.26\,\text{s}$, whereas Alice can verify her distance to about 470 peers per second.

1. Introduction

Location-based services have become integral in the digital age. The widespread use of geolocation-enabled devices, including smartphones and trackers like *Tile* and *AirTag*, has fueled advancements in services reliant on spatial data. However, these developments also raise significant privacy concerns, as location data reveals sensitive information about an individual's habits and preferences.

As a motivating example, consider digital contact tracing. Applications that collect geolocation records can, unintentionally or deliberately, expose sensitive user information, leading to potential privacy breaches. Moreover, these systems face the dual challenge of ensuring the privacy of honest users while preventing malicious actors from spoofing data to provide deliberately incorrect information.

In response to the first problem of protecting user location privacy, various Location Privacy Preserving Mechanisms (LPPM) protocols emerged [1], [2]. These solutions either apply *(i)* obfuscation [3], [4] or *(ii)* cryptographic methods [5], [6], [7], [8], where obfuscation reduces the precision of location data, while cryptographic approaches utilize secure computing and encryption to protect privacy. However, both solutions have shortcomings: differential privacy LPPM struggles with correlated user locations [9], and both cloaking [4] and Multi-Party Computation (MPC)-based cryptography [8] depend on third-party data anonymization.

Similarly, ensuring the authenticity of location information has gained importance in recent years, proliferating numerous solutions to mitigate spoofing [10]. For example, recent advancements in Global Navigation Satellite System (GNSS) address legacy satellite system vulnerabilities lacking signal authentication [11]. Additionally, efforts to inject fabricated location reports into offline finding networks like Apple's "Find My" are countered by manufacturers [12].

Considering these challenges, the ideal mechanism would allow clients to submit their own location proof to protect personalized location information without relying on third parties, it would preserve the utility of information with custom granularity, and it would prevent clients from deliberately providing wrong location information.

In this paper, we thus explore the following research question: *How can we obtain a short proof of location that allows for customized privacy preservation, while retaining accuracy and utility?*

We give an affirmative answer to our research question by introducing *Zero-Knowledge Location Privacy*. Our approach is driven by recent advancements in Zero-Knowledge Proofs (ZKPs), which enable practical use in applications previously deemed too costly. With ZKLP, users can prove to any third party that they are within a specific geographical region while obfuscating their exact location for utility and privacy. To do so, we rely on Discrete Global Grid Systems (DGGS) [13] with hexagons as geometric representation, which hierarchically divide the earth into progressively finer resolution grids. Any third party can only obtain an obfuscated location, whose correctness can be verified in milliseconds. To demonstrate the practicability of ZKLP, we develop a protocol for privacy-preserving peer-to-peer proximity testing and evaluate its practicality. While our main protocols focus on providing privacy to user-provided

§. Both authors contributed equally to this research.

locations, we introduce three possible solutions to ensure non-falsifiable location proofs in Appendix C.

Importance of Floating-Point Arithmetic for ZKLP. The necessity of floating-point arithmetic arises from the nature of data types and operations involved in geographic applications. Geolocation data (such as coordinates) and its associated computations (such as square roots $\sqrt{\cdot}$ and trigonometric functions $\sin(\cdot)$, $\cos(\cdot)$) are both in the domain of real numbers $\mathbb{R}$. In computers, these numbers and operations are typically represented using either fixed-point or floating-point formats. Generally, fixed-point arithmetic is more efficient and is thus commonly used in ZKP circuits [14], [15], while computer hardware and software often use floating-point arithmetic, which provides greater flexibility and supports a wider range of values.

For ZKLP, fixed-point arithmetic presents two key limitations. First, we need to simultaneously handle large values (e.g., $\sqrt{7}^{\text{res}}$, where the hexagonal resolution res can be up to 15) and small values (e.g., the product of $\sin\theta$ and $\cos\theta$) (cf. §4), resulting in either increased data size or reduced accuracy in fixed-point representation. In fact, compared with double precision floating point (FP64) that requires 64 bits, fixed-point costs nearly twice as many bits as FP64 while achieving lower accuracy (cf. §5.2). Second, our ZKLP circuits need to be compatible with existing geographic applications that are already implemented using IEEE 754 [16] floating-point arithmetic, such as Uber H3 [17]. The use of incompatible formats would compromise the completeness of ZKLP, since tiny turbulence in computations may yield significantly different results [18]. Looking ahead, in §5.2, with randomly generated test cases, fixed-point representation fails some tests whereas floating-point passes all. Even worse, this inconsistency can be exploited by adversaries to generate valid proofs for maliciously crafted statements, thereby breaking the soundness of ZKLP.

Challenges for Floating-Point. The major challenge in addressing our research question is achieving efficient floating-point arithmetic that is fully compliant with IEEE 754 in a Succinct Non-Interactive Argument of Knowledge (SNARK) circuit, which is nontrivial. A SNARK commonly operates over a finite field $\mathbb{F}_p$, where p is usually a large prime depending on the underlying elliptic curve (e.g., $|p| = 254$ for BN254). By contrast, floating-point numbers require integer operations in $\mathbb{Z}_{2^k}$ that are circuit-unfriendly, such as comparison and shifts.

Previous works have investigated the use of floating-point values in MPC [19], [20], [21] and ZKP [22], [23], [24]. However, they are either *(i)* not optimized for circuit size, *(ii)* inefficient in terms of communication cost and verification time, *(iii)* incapable of handling complex operations, or *(iv)* not fully compliant with the IEEE 754 standard [16] (see §6 for a detailed comparison).

For instance, [21], [22] naively convert software floating-point implementations to circuits, leading to a huge number of gates. Garg et. al [23] address this problem by proving the upper bound of the relative error, instead of transforming rounding operations on floating-point values to in-circuit bitwise operations. In spite of this, their work

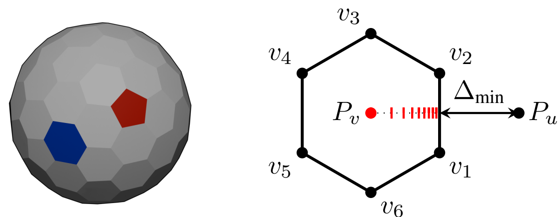


Figure 1: *(i)* Icosahedral Polyhedron. A hexagon is highlighted in blue, and a pentagon is highlighted in red. *(ii)* *Hexagon* $\mathbf{h} = [v_1, ..., v_6]$. P_u and P_v denote the points of verifier u and prover v respectively. Δ_{min} is the privacy-preserving distance of u to v. The red ticks depict our methodology to evaluate ZKLP in Section 5.

only supports addition and multiplication while lacking a concrete practical implementation. To realize ZKLP, we require complex operations such as division, taking square roots, and trigonometric functions. In addition, we also need to guarantee that these operations produce correctly rounded results. Hence, emulating floating-point numbers precisely, whilst attaining efficiency in the operations involved, is challenging and requires significant optimization.

Challenges for ZKLP. Transforming a location described as latitude and longitude to an index in a hexagonal grid system requires extensive use of trigonometric operations. Common approaches to approximating trigonometric functions by following the standard three-step recipe of *range reduction*, *polynomial approximation*, and *output compensation*, rely on high bitwidths to obtain precise results [25]. Further, naive polynomial approximation through, e.g., Taylor Series, is prohibitively expensive in-circuit, as precise results demand for many iterations, which is costly to represent in an arithmetic circuit that demands for linearization.

1.1. Contributions & Results

We provide a full implementation of IEEE 754 compliant floating-point operations in SNARKs and apply them in our implementation ZKLP. Our implementation of floating-point arithemtic is agnostic to the underlying SNARK arithemtization and applicable in orthogonal domains.

Our main technical contributions are *(i)* novel optimizations for computing floating-point SNARKs and *(ii)* optimizations to eliminate trigonometric operations that make the ZKLP paradigm practical. Throughout, we leverage lookup arguments and nondeterministic programming, enabling cost-effective representation of computations that are typically resource-intensive when executed in-circuit. To efficiently operate on floating-point values, we convert their integer components to an equivalent but circuit-efficient form, build optimized sub-circuits for integer operations, and minimize the number of costly range checks in key steps, e.g., rounding. For compliance with IEEE 754, we take additional care of edge cases, such as NaN, $\pm\infty$ and subnormal numbers. To efficiently instantiate ZKLP over primitive floating-point operations, we introduce shortcuts,

eliminating expensive math operations through trigonometric identities and nondeterministic programming.

Our experiments show that our circuits for primitive floating-point operations are precise and performant. We test our implementation with the Berkeley TestFloat library [26] to ensure full compliance. The extensive use of lookups leads to amortization — 2^1 single precision floating point (FP32) multiplications require 209 constraints, whereas 2^{15} FP32 multiplications require 64 constraints per operation. When applied to the ZKLP paradigm, our resulting circuits are highly efficient. In comparison to an unoptimized fixed-point baseline, our implementation has $15.9\times$ less constraints for FP32 values, and $12.2\times$ less constraints for FP64 values. We apply ZKLP and show that it can realize *privacy-preserving peer-to-peer proximity testing*, through which a user can evaluate its proximity to 470 peers per second.

In summary, our contributions are as follows:

- **ZKLP.** We introduce ZKLP, a novel application of ZKPs, along with a full implementation and evaluation. ZKLP unlocks a novel class of ZKP-empowered applications, which enable personalized location privacy through geo-indistinguishability. In particular, it allows individuals to prove that they have visited a certain location whilst ensuring privacy for the exact location.
- **IEEE 754 Compliant Floating-Point SNARKs.** We are the first to introduce optimized SNARK circuits for floating-point operations that are fully compliant to IEEE 754 [16] (§3). Our optimization for primitive floating-point operations are universally applicable, independent of specific arithmetizations, and fulfill the precision requirements of IEEE 754. We thus deem this contribution of independent interest.
- **Optimizations for ZKLP.** Transformations on geographic coordinates extensively demand for trigonometric functions. We introduce ZKP circuits that entirely eliminate trigonometric functions for ZKLP (§4).
- **Evaluation & Application.** We provide a full implementation and evaluation of all algorithms and optimizations, show the compliance of our floating-point circuits with IEEE 754, and showcase the ZKLP paradigm for peer-to-peer proximity testing (§5).

Limitations of ZKLP. Despite our optimizations making both floating-point and ZKLP circuits practical, there are notable limitations to our current evaluation. Specifically, our assessment does not factor in the additional overhead required to bridge the cyber-physical gap for obtaining provably authentic location information — a malicious prover could potentially falsify their location. To address this, we give three possible solutions to prevent the forgery of proofs attesting to incorrect locations in Appendix C, and argue about their estimated additional cost.

2. Preliminaries

2.1. Notation

We denote the bitwidth of $x \in \mathbb{Z}$ as $|x|$, and the absolute value as $\mathrm{abs}(x)$. $x \| y$ is the concatenation of $x, y \in \mathbb{Z}$. $x \ll n$ and $x \gg n$ shift x to the left and right by n bits, respectively, where $x \ll n = x \cdot 2^n$, $x \gg n = \lfloor \frac{x}{2^n} \rfloor$. For $x, y \in \mathbb{Z}$ with y's bitwidth known to be n, $\overline{x.y}$ is a shorthand for $\frac{x\|y}{2^n} \in \mathbb{R}$.

We define the position of a point on the Earth's surface in spacial coordinates by radial distance (r), latitude (θ), and longitude (ϕ). Here, r approximates the Earth's radius, θ represents the latitude, ranging from $-90°$ at the South Pole to $+90°$ at the North Pole with $0°$ at the Equator, and ϕ represents the longitude, ranging from $-180°$ to $+180°$ with $0°$ at the Prime Meridian. We represent a vertex of a hexagon in two-dimensional Cartesian coordinates as $v_i = (x_i, y_i)$. We denote a hexagon as a vector (in lower-case bold symbols) of 6 coordinates, i.e., $\mathbf{h} = [v_1, v_2, ..., v_6]$. Let $\mathcal{H} = \{\mathbf{h_1}, \mathbf{h_2}, ..., \mathbf{h_n}\}$ be a set of regular hexagons in the Euclidean plane, each hexagon $\mathbf{h_i}$ having a center point c_i and equal side length a. For each hexagon $\mathbf{h_i}$, there is a region R_i in the plane, defined by the vertices of $\mathbf{h_i}$, with the property that every point within R_i is closer to c_i than to the center point of any other hexagon in $\mathcal{H}$. The region R_i is referred to as a *hexagonal cell*.

2.2. Background On Floating-Point Values

Fixed-point and floating-point are common methods for representing real numbers in computers. In fixed-point representation, we fix a scaling factor N and represent a number x as an integer v of bitwidth $M > N$, whose lowest N bits are treated as the fractional part with an implicit decimal point in front, i.e., $x = \sum_{i=0}^{M-1} v_i \cdot 2^{i-N} = \frac{v}{2^N}$. Arithmetic operations on fixed-point numbers are equivalent to directly operating on their underlying integers, and only a small overhead is required to keep the scaling factor unchanged. The main advantage of fixed-point is that it can be more efficient in computation and storage, especially on low-cost hardware that lacks native support for floating-point arithmetic. However, it can only handle numbers with similar orders of magnitude, and lacks the ability to encode very large or very small values simultaneously and precisely.

Floating-point representation does not have a fixed scaling factor. It uses only a portion of the available bits (*mantissa*) to store the number, and the remaining bits (*exponent*) are reserved for dynamically tracking the scaling factor. When operating on floating-point numbers, the exponent and mantissa need to be correctly updated. While increasing cost, floating-point representation is more general than fixed-point representation, as it can represent very small and very large numbers with greater precision.

IEEE 754 [16] is the de facto standard for floating-point numbers and is widely adopted by modern hardware and software. It defines the encoding formats of floating-point numbers and a set of arithmetic operations on these numbers. A floating-point number in IEEE 754 consists of three components, the sign s, the exponent e, and the mantissa (or significand) m. The *binary* encoding format encodes these components as a binary string $s \| e \| m$, where s, e, m have $1, E, M$ bit(s) respectively. In this paper, we are interested in FP32, the single precision binary encoding

format with $E=8, M=23$, and FP64, the double precision one with $E=11, M=52$. Here, we briefly review the encoding itself and discuss arithmetic operations in §3.

The binary encoding format is capable of representing 5 types of values: *signed zeros* (± 0), *subnormal numbers*, *normal numbers*, *infinities* ($\pm\infty$), and *"not a number" values* (NaNs). We use *abnormal* to denote a number that is $\pm\infty$ or NaN. Below we list how the IEEE 754 specification maps the encoded value $s \parallel e \parallel m$ to the real number α.

(i) $e=0, m=0$ (± 0) — $\alpha=0$ if $s=0$ and $\alpha=-0$ if $s=1$.
(ii) $e=0, m\neq 0$ (subnormal) — $\alpha=(-1)^s \cdot 2^{-2^{E-1}+2} \cdot \overline{0.m}$.
(iii) $e \in [1, 2^E-2]$ (normal) — $\alpha=(-1)^s \cdot 2^{e-2^{E-1}+1} \cdot \overline{1.m}$. In this case, e is the *biased form* of the actual exponent with $2^{E-1}-1$ as the bias, and m, together with an implicit leading 1, describes the actual mantissa.
(iv) $e=2^E-1, m=0$ ($\pm\infty$) — $\alpha=\infty$ if $s=0$ and $\alpha=-\infty$ if $s=1$.
(v) $e=2^E-1, m\neq 0$ (NaN) — α isn't a numeric value.

As such, the encoding allows storing a normal number in the range $[2^{-2^{E-1}+2}, 2^{2^{E-1}})$ with $(M+1)$-bit precision and a subnormal number that is even smaller, i.e., in the range $[2^{-2^{E-1}+2-M}, 2^{-2^{E-1}+2})$, but with lower precision.

2.3. Discrete Global Grid Systems

A DGGS divides the Earth into a hierarchy of progressively finer resolution grids. Alternatively, a solution for dividing the Earth's surface would be to apply a simple latitude-longitude grid, where the Earth is divided into a grid based on lines of latitude and longitude, creating a series of rectangular cells that cover the entire globe. However, such an encoding leads to lines of longitude that are in closer proximity at the poles than they are at the equator. While a latitude-longitude grid is a simple way to partition the Earth's surface, it lacks the uniformity, efficiency, and scalability of a DGGS [13], particularly for complex spatial analyses and global-scale applications.

Hexagonal Hierarchical Geospatial Indexing. A long line of research suggests that defining a DGGS primarily based on hexagonal tiles exhibits superior properties than DGGS based on other geometric shapes for algorithmic efficiency [13], [27]. This benefit is evident in practical tools — Uber, for example, introduced the Hexagonal Hierarchical Spatial Index (H3), a geospatial index that partitions the globe into hexagons for more accurate analysis of movement patterns [17]. The global grid system relies on a gnomonic projection centered on icosahedron faces. In the Uber H3 indexing system, hexagons are utilized to create a grid on each icosahedron face. The H3 indexing system supports differing granularity, with 16 resolutions. At the highest resolution, 122 hexagons span the sphere of the earth, with 10 hexagons per icosahedron face. As hexagons cannot tile a icosahedron face, 12 pentagons are introduced at each of the icosahedron vertices to tile the full spherical projection. For enhanced intuition, Figure 1 depicts an Icosahedral Goldberg Polyhedron of hexagons and pentagons with 92 faces.

2.4. SNARKs

A zero-knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) [28] is a cryptographic protocol, where a prover $\mathcal{P}$ convinces a verifier $\mathcal{V}$ that a certain NP-statement is true, without disclosing any information besides the veracity of the statement. Common SNARKs target the problem of *circuit-satisfiability*, i.e., providing SNARKs for arbitrary NP-statements, represented as arithmetic circuits (R1CS [29], Plonkish [30], AIR [31]). Informally, a zk-SNARK for circuit satisfiability satisfies the following:

Succinct: The verification cost and the size of the proof are sublinear in the size of the circuit.

Non-Interactive: The prover $\mathcal{P}$ can provide a proof that can be independently verified without further communication.

Beyond the above properties, the security properties of a zk-SNARK can be informally described as follows:

Perfect Completeness: An honest $\mathcal{P}$ can always convince $\mathcal{V}$ of the correctness of a true statement.

Knowledge Soundness: A dishonest $\mathcal{P}$ cannot convince $\mathcal{V}$ of an invalid statement, except with negligible probability. Furthermore, an extractor can successfully extract the witness to a valid statement except with negligible probability.

Zero-Knowledge: The proof reveals nothing to $\mathcal{V}$ besides that $\mathcal{P}$ knows an assignment satisfying the circuit predicate.

We say an *argument of knowledge* retains knowledge soundness against a computationally bounded prover.

2.5. SNARK Optimizations

We introduce generic SNARK optimizations, applied in our protocols in Section 3 and 4, in the following.

Lookup Arguments. Most SNARKs can efficiently represent computations that can be expressed as an arithmetic circuit. However, there are some non-arithmetic operations, such as range checks, XOR or logical AND operations, that are unfriendly to the circuit and cost more constraints. Lookup arguments aim to reduce the prover complexity for these non-arithmetic operations by simply checking that a *query* is contained in a *lookup table* [32]. They were first introduced by Bootle *et. al* [33], and optimized in several successive works [34]. In this work, we consider read-only lookup tables. To build a lookup table $\mathcal{T}$, we precompute all valid values of a function and treat them as a vector of table entries $\mathbf{t} := (t_1, t_2, ..., t_n) \in \mathbb{F}^n$. Later, the prover is going to convince the verifier that a vector of queries $\mathbf{f} := (f_1, f_2, ..., f_m) \in \mathbb{F}^m$ is in the lookup table, i.e., $\mathbf{f} \subseteq \mathbf{t}$.

Based on logarithmic derivatives, LogUp [35] shows that it is sufficient to check the below identity for set inclusion:

$$\sum_{i=0}^{m-1} \frac{1}{X-f_i} = \sum_{j=0}^{n-1} \frac{o_j}{X-t_j},$$

where o_j is the number of t_j's occurrences in the query vector $\mathbf{f}$. By Schwartz-Zippel Lemma, we can check this polynomial identity by evaluating it at a random point $X=c$. While LogUp [35] provides a dedicated protocol for this check, we instead adopt the approach in gnark [36], which

enforces the identity in arithmetic circuits. Also, note that the identity can be extended to support lookup tables with $w > 1$ columns, where each element in the queries and entries is now a vector of length w (i.e., $f_i, t_i \in \mathbb{F}^w$).

Nondeterministic Programming. An in-circuit computation proves that the input data satisfies a given compliance predicate. The local input data can provide arbitrary *hints* (or nondeterministic advice [37]), which are not trusted to be correct, but whose verification is more efficient in-circuit than the emulation of the plain computation. Hints leverage the fact that certain calculations are hard to compute, but easy to verify in an arithmetic circuit.

In consonance with related work [38], we formalize a *hint* as the computation $H(X) \to Y$ done by the prover outside the arithmetic circuit. An in-circuit nondeterministic predicate $P : X \times Y \to \{0, 1\}$ for H ensures that $\forall x \in X, y \in Y$ the relations $H(x) = y \iff P(x, y) = 1$ and $H(x) \neq y \iff P(x, y) = 0$ hold. Note, that in practical applications, the variable returned by a hint function is equivalent to a prover-supplied witness. We call those variables hints instead of witnesses for separation of concerns, highlighting that the computation is done outside the circuit, and that their values are provided by the prover.

For example, consider extracting the most significant bit of the mantissa of a floating point value. Let the computation to find the MSB of a mantissa be $H_{\mathsf{MSB}}(m) = \mathrm{MSB}_m$ and the nondeterminism predicate be $P_{\mathsf{MSB}} : \mathbb{Z}_p \times \mathbb{Z}_p \to \{0, 1\}$. In the circuit, it is verified that MSB_m is indeed the MSB of m by the predicate P_{MSB}:

$$P_{\mathsf{MSB}}(m, \mathrm{MSB}_m) = \begin{cases} 1 & \text{if } m - \mathrm{MSB}_m \cdot 2^{|m|-1} \in [0, 2^{m-1}], \\ 0 & \text{otherwise.} \end{cases}$$

If $\mathrm{MSB}_m = 0$ but the actual MSB_m is 1, then the expression $m - \mathrm{MSB}_m \cdot 2^{|m|-1}$ will result in at least $|m|$ bits. If $\mathrm{MSB}_m = 1$ but the actual MSB is 0, the subtraction will result in a negative value.

3. Primitive Floating-Point Operations

In the following, we introduce optimized floating-point circuits defined over a prime field $\mathbb{F}_p$ of order p for primitive operations (addition, subtraction, multiplication, division, square root, and comparison). Here, we denote integer-typed variables as Latin letters $(a, b, \dots)$, and floating-point values as Greek letters $(\alpha, \beta, \dots)$. To avoid verbosity, we omit the in-circuit constructions for equality constraint $(x = y)$, equality check $(result \coloneqq \mathcal{C}_{\mathsf{IsEq}}(x, y))$, conditional selection $(condition\,?\,true_value : false_value)$, and boolean operations $(\wedge, \vee, \neg, \oplus)$, etc. Further, we introduce circuits for integer operations in Appendix A, including $\mathcal{C}_{\mathsf{RC}}$ for range checks, $\mathcal{C}_{\mathsf{Abs}}$ for extracting signs and absolute values, $\mathcal{C}_{\mathsf{Max}}$ and $\mathcal{C}_{\mathsf{Min}}$ for computing maximum and minimum values, and $\ll$ and $\gg$ for left and right shifting.

3.1. Initializing Floating-Point Numbers

Now we discuss how to initialize a floating-point number α inside the circuit, whose original representation is $\alpha = (\hat{s}, \hat{e}, \hat{m})$, where $\hat{s} \in \{0, 1\}$ is the sign bit, $\hat{e}$ is an E-bit exponent, and $\hat{m}$ is an M-bit mantissa (or significand). Note that although it is possible to represent α in circuit as-is, we convert it to a compatible format $\alpha = (s, e, m, a)$ to save as much constraints as possible, where s, e, m are circuit-efficient form of $\hat{s}, \hat{e}, \hat{m}$, and $a \in \{0, 1\}$ is an additional bit indicating whether the number is abnormal.

$\mathcal{C}_{\mathsf{FpInit}}(\hat{s}, \hat{e}, \hat{m})$

1: $\hat{s}(1 - \hat{s}) = 0; \mathcal{C}_{\mathsf{RC}}(\hat{e}, E); \mathcal{C}_{\mathsf{RC}}(\hat{m}, M)$
2: $s \coloneqq \hat{s}; e \coloneqq \hat{e}; m \coloneqq \hat{m}$
3: $m_is_0 \coloneqq \mathcal{C}_{\mathsf{IsEq}}(m, 0)$
4: $e_is_min \coloneqq \mathcal{C}_{\mathsf{IsEq}}(e, 0); e_is_max \coloneqq \mathcal{C}_{\mathsf{IsEq}}(e, 2^E - 1)$
5: Receive hint $d = H_{\mathsf{Norm}}(m)$
6: $\mathcal{C}_{\mathsf{RC}}((m \ll d) - (m_is_0\,?\,0 : 2^M), M)$
7: $e \coloneqq -2^{E-1} + 1 + (e_is_min\,?\,(m_is_0\,?\,-M : 1 - d) : e)$
8: $m \coloneqq (e_is_max \wedge \neg m_is_0)?0 : (e_is_min?m \ll d : m + 2^M)$
9: $a \coloneqq e_is_max$

Figure 2: Circuit for initializing floating-point numbers

Shape Check. On unchecked input $\hat{s}$, $\hat{e}$, and $\hat{m}$ (which are usually secret witnesses), we first enforce that $\hat{s}$, $\hat{e}$ and $\hat{m}$ are well-formed in-circuit by checking $\hat{s} \in \{0, 1\}, \hat{e} \in [0, 2^E - 1], \hat{m} \in [0, 2^M - 1]$. After that, we initialize $s \coloneqq \hat{s}, e \coloneqq \hat{e}, m \coloneqq \hat{m}$.

Convert to Circuit Efficient Forms. For a normal number $\alpha = (-1)^s \cdot 2^{e - 2^{E-1} + 1} \cdot \overline{1.m}$, the sign s is unchanged, while we redefine e as the exponent's unbiased form, i.e., $e \coloneqq e - 2^{E-1} + 1$, and m as the mantissa with an explicit leading bit 1, i.e., $m \coloneqq m + 2^M$. In this way, we no longer need to handle the bias of e and the implicit leading bit of m in subsequent operations, thus improving efficiency and clarity.

We convert subnormal numbers to normal numbers by normalizing their mantissas, allowing the exponents to 'underflow'. Specifically, for a subnormal number $\alpha = (-1)^s \cdot 2^{-2^{E-1}+2} \cdot (0 \parallel m) \cdot 2^{-M}$, we left shift the mantissa (with leading zero) $0 \parallel m$, and at the same time decrement the exponent $-2^{E-1} + 2$ by 1, until the MSB (i.e., M-th bit) of mantissa becomes 1. Denoting the shift by $d \in [1, M]$, we have $\alpha = (-1)^s \cdot 2^{-2^{E-1}+2-d} \cdot ((0 \parallel m) \ll d) \cdot 2^{-M}$ and redefine $e \coloneqq -2^{E-1} + 2 - d, m \coloneqq (0 \parallel m) \ll d = m \ll d$. Now, the prover computes the hint $d \coloneqq H_{\mathsf{Norm}}(m)$ and provides d to the circuit. For soundness, the circuit needs to check the predicate $P_{\mathsf{Norm}}(m, d)$ by calling $\mathcal{C}_{\mathsf{RC}}((m \ll d) - (m_is_0\,?\,0 : 2^M), M)$, which enforces m is zero or $m \ll d \in [2^M, 2^{M+1} - 1]$, implying that the MSB (i.e., M-th bit) of a non-zero $m \ll d$ is 1.

Normalizing unifies normal and subnormal numbers to save constraints in subsequent operations. Although subnormal numbers now take $M + 1$ bits to represent, padding zeros do not contribute to precision, so they still have lower precision than normal numbers.

Edge Cases. $\pm\infty$ is represented by $e = 2^{E-1}, m = 2^M$. ± 0 is represented by $e = -2^{E-1} + 1 - M, m = 0$. Although NaNs have different mantissas as per the specification, we always map them to fixed variables $s = 0, e = 2^{E-1}, m = 0$ to simplify the handling of edge cases. We set 0 as NaN's mantissa due to the similar behaviors between NaNs and ± 0

$\mathcal{C}_{\mathsf{FpRound}}(e, m, \Delta e, aux = 1)$

Require: $m \in [0, 2^N - 1], \Delta e \in [0, K], 2^{N+K} < p$
1: Receive hints $u', b_1, b_2, v' = H_{\mathsf{Split}}(m \ll (K - \Delta e))$
2: $\mathcal{C}_{\mathsf{RC}}(u', M); b_1(1-b_1) = 0; b_2(1-b_2) = 0; \mathcal{C}_{\mathsf{RC}}(v', N-M-2+K)$
3: $m \ll K = (u' \parallel b_1 \parallel b_2 \parallel v') \ll \Delta e$
4: $u := u' \parallel b_1; v := b_2 \parallel v'$
5: $half := \mathcal{C}_{\mathsf{IsEq}}(v, 2^{N-M-2+K}) \wedge aux$
6: $m' := (u + (half\ ?\ b_1 : b_2)) \ll \Delta e$
7: $overflow := \mathcal{C}_{\mathsf{IsEq}}(m', 2^{M+1})$
8: $e' := e + overflow$
9: $m' := overflow\ ?\ 2^M : m'$
10: **return** e', m'

Figure 3: Circuit for rounding floating-point numbers

in multiplication and division.

We compute the final exponent and mantissa inside the circuit using several conditional selections. The entire circuit for initializing floating-point variables is shown in Figure 2.

3.2. Rounding

The IEEE 754 standard requires the resulting mantissa of an operation to be rounded correctly and defines several rules specifying how and in which direction the rounding is done. Here, we discuss "rounding half to even" for binary encoded floating-point numbers. This rule requires a decimal number to be rounded to the nearest integer, except when the fractional part is 0.5 (in base-10), the rounding should produce an even value, e.g., $0.5 \to 0, 1.5 \to 2$.

Formally, consider an intermediate mantissa $m \in [0, 2^N - 1]$ after some operation, and we are going to compute a rounded value $m' \in [0, 2^l - 1]$. To this end, we write m as $m = u \| v$, where $u \in [0, 2^l - 1]$ and $v \in [0, 2^{N-l} - 1]$. The rounding direction is determined by the bits in v, which are the *round bit*, the *guard bit*, and the *sticky bit* in most implementations of IEEE 754. In our circuits, rounding is done according to only the round bit and sticky bit, where the round bit is v_{N-l-1} (the MSB of v), and the sticky bit is $v_0 \vee \cdots \vee v_{N-l-2}$ (the OR value of remaining bits of v). The guard bit is useful for implementations that discard the right-shifted bits but is unnecessary in our case because these bits are preserved. Consequently, we have $m' := u$ if round bit is 0, i.e., $v \in [0, 2^{N-l-1})$. If round bit is 1 and sticky bit is 0, i.e., $v = 2^{N-l-1}$, then $m' := u + 1$ when u is odd, while $m' := u$ when u is even. Otherwise, $v \in (2^{N-l-1}, 2^{N-l})$, and the result $m' := u + 1$.

Note that due to the possible increment in $m' := u + 1$, m' could become 2^l, exceeding the upper bound $2^l - 1$. We say m' overflows in this case, and we fix this by setting $e' := e + 1, m' := 2^{l-1}$.

If we follow the standard exactly, l will be fixed to $M+1$. However, in our case, $l = M + 1$ only when the result is normal. For a subnormal number with an underflown exponent $e < -2^{E-1} + 2$, we require $l = M + 1 - \Delta e$, where $\Delta e = -2^{E-1} + 2 - e$ (here it is guaranteed that $e \geq -2^{E-1} + 1 - M$, implying $l \geq 0$). The reason is that, for circuit efficiency, we pretend that subnormal numbers are normal when representing and operating on them. This works in most cases except for rounding, where subnormal numbers should actually be rounded with lower precision (i.e., $M+1-\Delta e$) than that of normal numbers (i.e., $M+1$). Hence, we should only keep the first $M + 1 - \Delta e$ bits of m for subnormal numbers. When the rounding is done, we left shift m' by Δe to continue disguising them as normal.

Figure 3 illustrates the rounding gadget, where we require that Δe has a constant upper bound K, and that $\Delta e = 0$ for normal numbers. First, we need to expand m into $u \parallel v$ in-circuit. Naively, we can ask the prover to provide u, v as hints, and the circuit checks $u \in [0, 2^l - 1], v \in [0, 2^{N-l} - 1]$. However, the upper bounds of u, v depend on the variable l. As discussed in Appendix A, a range check bounded by variables costs more constraints than a constant range check. To maximize efficiency, the prover instead computes the hint H_{Split} by expanding $m \ll (K - \Delta e)$ into $u' \parallel b_1 \parallel b_2 \parallel v'$, where $u' \in [0, 2^{M-\Delta e} - 1], b_1, b_2 \in \{0, 1\}$ and $v' \in [0, 2^{N-M-2+K} - 1]$. Then $u', b_1, b_2, v' := H_{\mathsf{Split}}(m \ll (K - \Delta e))$ are fed to the circuit. Now, $u := u' \| b_1$, $v := b_2 \| v'$. To verify the predicate $P_{\mathsf{Split}}(m \ll (K - \Delta e), u', b_1, b_2, v')$, the circuit checks the ranges of u' and v' by calling $\mathcal{C}_{\mathsf{RC}}$, enforces b_1 and b_2 are boolean, and asserts $m \ll K = (u' \parallel b_1 \parallel b_2 \parallel v') \ll \Delta e$. Moreover, as m is guaranteed to have length N, we can loosely bound u' and check $u' \in [0, 2^M - 1]$ instead. This is safe, since if $2^{M-\Delta e} \leq u' < 2^M$, the length of $u' \| b_1 \| b_2 \| v'$ will be longer than $N + K - \Delta e$, and m's length will be longer than N, which is a contradiction. Now, both range checks are bounded by constants.

After that, we compute $half := \mathcal{C}_{\mathsf{IsEq}}(v, 2^{N-M-2+K}) \wedge aux$, where aux is the auxiliary information that helps determine the sticky bit in division and the computation of square root. For $aux = 1$, the sticky bit solely depends on v. Then we have the rounded mantissa $m' := (u + (half\ ?\ b_1 : b_2)) \ll \Delta e$. Finally, we check $overflow := \mathcal{C}_{\mathsf{IsEq}}(m', 2^{M+1})$, and return the updated exponent and mantissa $e' := e + overflow, m' := overflow\ ?\ 2^M : m'$.

3.3. Addition And Subtraction

Adding two IEEE 754 floating-point numbers $\alpha = (s_\alpha, e_\alpha, m_\alpha, a_\alpha)$ and $\beta = (s_\beta, e_\beta, m_\beta, a_\beta)$ is done in the following 5 steps, and we depict the corresponding in-circuit logic in Figure 4. At a high level, addition requires 5 steps, described in the following: *(i)* aligning exponents, *(ii)* adding mantissas, *(iii)* normalizing, *(iv)* rounding the intermediate mantissa and *(v)* handling edge cases. Note, that subtraction is equivalent to addition by adding α and $-\beta$.

Align exponents (lines 1-3). We first compare the exponents of α and β. If $e_\alpha \neq e_\beta$, we need to align the exponents before performing the actual addition by shifting the mantissa of the number with smaller exponent to the *right* by $abs := \mathrm{abs}(e_\alpha - e_\beta)$ bits, such that the common exponent is $e := \max(e_\alpha, e_\beta)$. To avoid separately tracking the shifted bits (which will be used later in rounding), before performing the right shift, we first shift both mantissas to the *left* by L bits, where L is the upper bound of abs. That is, if $e_\alpha > e_\beta$, we compute $x := m_\alpha \ll L, y := (m_\beta \ll L) \gg (e_\alpha - e_\beta)$,

$\underline{\alpha + \beta}$

1: $c, abs \coloneqq \mathcal{C}_{\mathsf{Abs}}(e_\beta - e_\alpha, E+1)$
2: $e \coloneqq c \,?\, e_\beta : e_\alpha$
3: $abs \coloneqq \mathcal{C}_{\mathsf{Min}}(abs, M+3)$
4: $x \coloneqq (c \,?\, s_\beta m_\beta : s_\alpha m_\alpha) \ll L$
5: $y \coloneqq (c \,?\, s_\alpha m_\alpha : s_\beta m_\beta) \ll (L - abs)$
6: $z \coloneqq x + y$
7: $\neg s, m \coloneqq \mathcal{C}_{\mathsf{Abs}}(z, 2M+5)$
8: $e \coloneqq e + 1$
9: $a \coloneqq a_\alpha \vee a_\beta$
10: $m_is_0 \coloneqq \mathcal{C}_{\mathsf{IsEq}}(m, 0)$
11: Receive hint $H_{\mathsf{Norm}}(v) = d$
12: $m \coloneqq m \ll d; e \coloneqq e - d$
13: $\mathcal{C}_{\mathsf{RC}}(m - (m_is_0 \,?\, 0 : 2^{2M+4}), 2M+4)$
14: $e', m' \coloneqq \mathcal{C}_{\mathsf{FpRound}}(e, m, 0)$
15: $a' \coloneqq a \vee \mathcal{C}_{\mathsf{GEZ}}(e' - 2^{E-1}, E+1)$
16: $s' \coloneqq \mathcal{C}_{\mathsf{IsEq}}(s_\alpha, s_\beta) \,?\, s_\alpha : s$
17: $e' \coloneqq a' \,?\, 2^{E-1} : (m_is_0 \,?\, -2^{E-1} + 1 - M : e')$
18: $m'_1 \coloneqq (\neg a_\beta \vee \mathcal{C}_{\mathsf{IsEq}}(s_\alpha m_\alpha, s_\beta m_\beta)) \,?\, m_\alpha : 0$
19: $m'_2 \coloneqq a_\beta \,?\, m_\beta : (a' \,?\, 2^M : m')$
20: $m' \coloneqq a_\alpha \,?\, m'_1 : m'_2$
21: **return** s', e', m', a'

Figure 4: Circuit for floating-point addition

and otherwise, $x \coloneqq m_\beta \ll L, y \coloneqq (m_\alpha \ll L) \gg (e_\beta - e_\alpha)$. The $M+1$ MSB's of shifted mantissas contribute to the final result, the remaining bits determine the rounding direction.

In circuit, we achieve this by computing $c, abs \coloneqq \mathcal{C}_{\mathsf{Abs}}(e_\beta - e_\alpha, E+1)$ to obtain $e \coloneqq c \,?\, e_\beta : e_\alpha$. We observe that the final sum is only determined by x when y is completely shifted out, i.e., when $abs \geq M+3$. Thus, $abs > M+3$ has the same effect as $abs = M+3$. We improve the circuit efficiency by setting $abs \coloneqq \mathcal{C}_{\mathsf{Min}}(abs, M+3)$, so that it is no longer necessary to compute a large 2^{abs}. Now, $L = M+3$, $x \coloneqq (c \,?\, m_\beta : m_\alpha) \ll L, y \coloneqq (c \,?\, m_\alpha : m_\beta) \ll (L - abs)$.

Add signed mantissas (lines 4-9). Then we add the signed mantissas of adjusted α and β, and obtain the resulting (signed) mantissa, i.e., $s \cdot m \coloneqq s_\alpha \cdot x + s_\beta \cdot y$ if $e_\alpha > e_\beta$, and $s \cdot m \coloneqq s_\beta \cdot x + s_\alpha \cdot y$ otherwise. The sign s and unsigned mantissa m are extracted from the result, where $m \leq x + y < 2(2^{M+1} \cdot 2^L) = 2^{2M+5}$, i.e., m has at most $N = 2M+5$ bits, where the leading bit is caused by the possible carry. Thus, we also adjust the exponent as $e \coloneqq e + 1$. For efficiency, we redefine the in-circuit variables $x \coloneqq (c \,?\, s_\beta m_\beta : s_\alpha m_\alpha) \ll L, y \coloneqq (c \,?\, s_\alpha m_\alpha : s_\beta m_\beta) \ll (L - abs)$ to avoid extra conditional selections. Then we compute $z \coloneqq x + y$, and compute s and m thanks to the $\mathcal{C}_{\mathsf{Abs}}$ gadget: $\neg s, m \coloneqq \mathcal{C}_{\mathsf{Abs}}(z, N)$. The result is abnormal if either input is abnormal, i.e., $a \coloneqq a_\alpha \vee a_\beta$.

Normalize intermediate mantissa (lines 10-13). Normalization for m is the same as in Section 3.1 for normalizing subnormal numbers. m is shifted to the left by d bits, so that its MSB (i.e., the $N-1$-th bit) becomes 1, unless $m = 0$, and e is decreased by d.

Round intermediate mantissa (line 14). The normalized mantissa $m \ll d$ of length N is then rounded as in Section 3.2, with $\Delta e = K = 0$, obtaining e' and m'. Theoretically, Δe should be $-2^{E-1} + 2 - e$. However, we observe that for addition, a smaller Δe doesn't affect the result. In fact, it is safe to set $\Delta e = 0$ to maximize circuit efficiency, and we explain the reasoning below.

Recall that the purpose of Δe is to limit the precision of subnormal numbers, but, as we will show later, the number of meaningful bits in $m \ll d$ will never exceed the required precision, and the remaining bits are guaranteed to be zero. Consider a subnormal result with $e \leq -2^{E-1} + 1$, and assume, without loss of generality, $e_\alpha < e_\beta$. We denote $\Delta e_\alpha = -2^{E-1} + 2 - e_\alpha, \Delta e_\beta = -2^{E-1} + 2 - e_\beta$. Since $e = \max(e_\alpha, e_\beta) + 1 - d = e_\beta + 1 - d$, we have $e_\alpha < e_\beta \leq -2^{E-1} + d$. According to the rounding rule, we need to keep only $M + 1 - (-2^{E-1} + 2 - e)$ bits in the mantissa, while the remaining $N - (M+1) + (-2^{E-1} + 2 - e) = L + d + \Delta e_\beta$ bits determine the rounding direction. Now we prove that the number of trailing zeros in $m \ll d$ is at least $L + d + \Delta e_\beta$ if $e \leq -2^{E-1} + 1$. Note, that $x = s_\beta m_\beta \ll L, y = s_\alpha m_\alpha \ll (L - \min(e_\beta - e_\alpha, L)) = s_\alpha m_\alpha \ll \max(L - e_\beta + e_\alpha, 0)$.

(i) *α subnormal, β subnormal* — In this case, m_α and m_β were left-shifted by Δe_α and Δe_β bits when initialized. Also, $e_\beta - e_\alpha \leq -2^{E-1} + 1 - (-2^{E-1} + 1 - M) = M < L$, thus $\max(L - e_\beta + e_\alpha, 0) = L - e_\beta + e_\alpha$. Now, x has at least $\Delta e_\beta + L$ trailing zeros, and y has at least $\Delta e_\alpha + \max(L - e_\beta + e_\alpha, 0) = \Delta e_\alpha + L - e_\beta + e_\alpha = \Delta e_\beta + L$ trailing zeros. Hence, $m \ll d$ has at least $\Delta e_\beta + L + d$ trailing zeros.

(ii) *α subnormal, β normal* — Here, m_α was left-shifted by Δe_α bits when initialized. Now, x has at least L trailing zeros, and y has at least $\Delta e_\alpha + \max(L - e_\beta + e_\alpha, 0) = \max(L + \Delta e_\beta, \Delta e_\alpha) < L$ trailing zeros. So $m \ll d$ has at least $\max(L + \Delta e_\beta, \Delta e_\alpha) + d \geq L + \Delta e_\beta + d$ trailing zeros.

(iii) *α normal, β normal* — In this case, $e_\beta - e_\alpha \leq -2^{E-1} + d - (-2^{E-1} + 2) = d - 2 < L$, thus $\max(L - e_\beta + e_\alpha, 0) = L - e_\beta + e_\alpha$. Now, x has at least L trailing zeros, and y has at least $\max(L - e_\beta + e_\alpha, 0) = L - e_\beta + e_\alpha < L$ trailing zeros. Consequently, $m \ll d$ has at least $L - e_\beta + e_\alpha + d$ trailing zeros, and $L - e_\beta + e_\alpha + d \geq L + d + \Delta e_\beta$ because $\Delta e_\beta + e_\beta - e_\alpha = \Delta e_\alpha \leq 0$.

Edge Cases (lines 15-21). Finally, we need to handle the following:

(i) If the mantissa $m' = 0$ but $e' \neq -2^{E-1} + 1 - M$ (which is possible, e.g., when computing $1.0 - 1.0$), canonicalize the exponent as $e' \coloneqq -2^{E-1} + 1 - M$.
(ii) If the exponent becomes too large, i.e., $e' \geq 2^{E-1}$, return $\pm\infty$ (depending on the sign s).
(iii) If $\alpha = \beta = \pm 0$, return -0 for $-0 - 0$ and $+0$ otherwise.
(iv) If either α or β is NaN, return NaN.
(v) If either α or β is $\pm\infty$, return NaN for $\infty - \infty$ and $-\infty + \infty$, and otherwise, $\pm\infty$ (depending on s).
(vi) Otherwise, return $(s, e', m', 0)$ as the result.

To minimize the number of constraints, we unify some cases above based on the return value's exponent and mantissa in-circuit. First, the result is abnormal if either inputs is abnormal (*iv*, *v*), or the exponent is too large (*ii*). Hence, $a' \coloneqq a \vee \mathcal{C}_{\mathsf{GEZ}}(e' - 2^{E-1}, E+1)$. Second, to support *(iii)*, we set $s' \coloneqq \mathcal{C}_{\mathsf{IsEq}}(s_\alpha, s_\beta) \,?\, s_\alpha : s$. Third, the result's exponent is 2^{E-1} if the result is abnormal (*ii*, *iv*, *v*), and is $-2^{E-1} + 1 - M$ if the result is zero (*i*). Thus, $e' \coloneqq a' \,?\, 2^{E-1} : (m_is_0 \,?\, -2^{E-1} + 1 - M : e')$. Finally, for the result's mantissa m', if both α and β are abnormal, $m' = 2^M$ for

```
α · β
1: s := s_α ⊕ s_β; e := e_α + e_β; m := m_α · m_β; a := a_α ∨ a_β
2: Receive hint b = H_MSB(m)
3: b(1 − b) = 1; C_RC(m − (b ≪ (2M + 1)), 2M + 1)
4: m := b ? m : m ≪ 1; e := e + b
5: Δe := C_Max(C_Min(−2^{E−1} + 2 − e, M + 2, E + 1), E + 1)
6: e′, m′ := C_FpRound(e, m, Δe)
7: a′ := a ∨ C_GEZ(e′ − 2^{E−1}, E + 1)
8: m′_is_0 := C_IsEq(m′, 0)
9: e′ := a′ ? 2^{E−1} : (m′_is_0 ? −2^{E−1} + 1 − M : e′)
10: m′ := (a′ ∧ ¬m′_is_0) ? 2^M : m′
11: return s, e′, m′, a′
```

Figure 5: Circuit for floating-point multiplication

$\infty + \infty$ and $-\infty - \infty$ (*v*), and $m' = 0$ otherwise (*iv*, *v*). If only one of α and β is abnormal, m' equals the abnormal one's mantissa (*iv*, *v*). If both α and β are normal, the result is 2^M if the exponent becomes too large (*ii*). Lines 19-21 in Figure 4 summarize the logic above for handling m'.

3.4. Multiplication And Division

Multiplying two IEEE 754 floating-point numbers $\alpha = (s_\alpha, e_\alpha, m_\alpha, a_\alpha)$ and $\beta = (s_\beta, e_\beta, m_\beta, a_\beta)$ is done in the following 4 steps — *(i)* computing the product of α and β, *(ii)* normalizing and *(iii)* rounding the intermediate mantissa and *(iv)* handling edge cases. We depict the corresponding in-circuit logic in Figure 5. The steps of division operation are highly similar to those of multiplication, and we defer their description to Appendix B due to the space limit.

Compute product (line 1). The product is negative only when one of α and β is negative. Therefore, the sign of the product is $s := s_\alpha \oplus s_\beta$. The exponent and mantissa of the product are respectively $e := e_\alpha + e_\beta$ and $m := m_\alpha \cdot m_\beta$. Since m_α, m_β are either 0 or lie in $[2^M, 2^{M+1}-1]$, a non-zero m should be bounded by $m \in [2^{2M}, 2^{2M+2})$. We further compute $a := a_\alpha \vee a_\beta$.

Normalize intermediate mantissa (lines 2-4). By leveraging the fact that m is either 0 or in $[2^{2M}, 2^{2M+2})$, the leading 1 of a non-zero m is either the $2M$-th bit or the $2M+1$-th bit. Hence, we can simplify the normalization process by checking if the $2M+1$-th bit of m is 1. If this is the case, m is already normal, and otherwise, we compute $m := m \ll 1$. Also, $m_{2M+1} = 1$ indicates that the multiplication carries, and hence we increment $e := e+1$ if so. The improved normalization is done in-circuit as follows: the prover feeds $b := H_{\mathsf{MSB}}(m) = m_{2M+1}$, the MSB of m, as a hint to circuit, and the circuit checks the predicate $P_{\mathsf{MSB}}(m, b)$ in 2 steps: *(i)* enforce b is a boolean, and *(ii)* assert $m - (b \ll (2M+1)) \in [0, 2^{2M+1})$. Then m, e are updated according to b, i.e., $m := b \,?\, m : m \ll 1$, $e := e + b$.

Round intermediate mantissa (lines 5-6). The normalized mantissa m of length $N = 2M+2$ is rounded by following the steps in Section 3.2, with $\Delta e = \max(\min(-2^{E-1}+2-e, K), 0), K = M+2$, obtaining e' and m'.

Edge Cases (lines 7-11). Finally, we handle the following:

(i) If the exponent becomes too large, i.e., $e' \geq 2^{E-1}$, return $\pm\infty$ (depending on the sign s).

(ii) If the exponent becomes too small, i.e., $e' < -2^{E-1} + 1 - M$, or equivalently, the rounded mantissa becomes 0, return ± 0 (depending on the sign s).

(iii) If either α or β is NaN, return NaN.

(iv) If either α or β is $\pm\infty$, return NaN for $\pm 0 \cdot \pm\infty$ and $\pm\infty \cdot \pm 0$, and otherwise, $\pm\infty$ (depending on s).

(v) Otherwise, return $(s, e', m', 0)$ as the result.

To minimize the number of constraint, we unify some cases above based on the return value's exponent and mantissa in-circuit. First, the result is abnormal if either input is abnormal (*iii*, *iv*), or the exponent is too large (*i*). Hence, $a' := a \vee \mathcal{C}_{\mathsf{GEZ}}(e' - 2^{E-1}, E+1)$. Second, the result's exponent is 2^{E-1} if the result is abnormal (*i*, *iii*, *iv*), and is $-2^{E-1} + 1 - M$ if the result is zero (*ii*). Thus, $e' := a' \,?\, 2^{E-1} : (m'_is_0 \,?\, -2^{E-1}+1-M : e')$. Finally, the result's mantissa is only different from the rounded mantissa if the exponent is too large (*i*), or either inputs is $\pm\infty$ and neither of them is ± 0 (*iv*). Both conditions are equivalent to the case where the result is abnormal but not NaN, so we have $m' := (a' \wedge \neg m'_is_0) \,?\, 2^M : m'$.

```
√α
1: s := s_α
2: Receive hint b = H_LSB(e_α)
3: e := (e_α − b)/2
4: b(1 − b) = 1; C_Abs(e, E − 1)
5: Receive hint n = H_Sqrt(m_α ≪ (M + 4 + b))
6: r := (m_α ≪ (M + 4 + b)) − n^2
7: C_RC(r, M + 4), C_RC(2n − r, M + 4)
8: m := n
9: m_is_0 := C_IsEq(m_α, 0)
10: a := a_α ∨ (s_α ∧ ¬m_is_0)
11: e′, m′ := C_FpRound(e, m, 0, C_IsEq(r, 0))
12: e′ := a ? 2^{E−1} : (m_is_0 ? −2^{E−1} + 1 − M : e′)
13: m′ := s ? 0 : m′
14: return s, e′, m′, a
```

Figure 6: Circuit for floating-point square root

3.5. Square Root Computation

The approximation of square roots is often based on the iterative Newton method [39]. To compute $\beta = \sqrt{\alpha}$, we first estimate an initial value β_0, and improve the accuracy in each round by computing $\beta_{i+1} = \frac{1}{2}(\beta_i + \frac{\alpha}{\beta_i})$. However, directly translating this approach to in-circuit operations introduces two challenges: *(i)* the number of iterations depends on how fast $\sqrt{\alpha}$ converges, but handling loops conditioned on a variable in-circuit is hard, and *(ii)* each round of iteration requires one floating-point addition and one floating-point division, which are costly. To address *(i)*, we need to run the loop for fixed number of rounds, taking the worst case for convergence into account, e.g., achieving the accuracy of FP64 needs 6 rounds of iteration in the worst case. We can resolve *(ii)* by computing the square root of the mantissa m_α rather than the floating-point value α. β's exponent can be obtained by halving e_α. Since m_α is an integer, addition and division in each round are cheap.

Nevertheless, this improved approach would easily cost hundreds of constraints due to the range checks caused by

in-circuit integer division. To further reduce circuit size, we leverage the nondeterminism of the constraint system: the prover is asked to compute the square root of m_α outside the circuit, and the circuit, given the square root as a hint, only needs to check its validity, thereby achieving the minimum cost. More specifically, we compute the square root of an IEEE 754 floating-point number $\alpha = (s_\alpha, e_\alpha, m_\alpha, a_\alpha)$ inside the circuit in the following 4 steps, the process of which is also shown in Figure 6.

Compute square root (lines 1-10). First, we halve the exponent e_α. When e_α is even, we can simply compute $e_\alpha/2$, and otherwise, we need to calculate $(e_\alpha - 1)/2$. Combining both cases, the prover feeds $b \coloneqq H_{\mathsf{LSB}}(e_\alpha)$, the exponent's LSB, as a hint to circuit. The circuit checks the predicate $P_{\mathsf{LSB}}(e_\alpha, b)$ as follows: enforce b is boolean, compute $e \coloneqq (e_\alpha - b)/2$, and assert $e \in [-2^{E-1} + 1, 2^{E-1} - 1]$ by calling $\mathcal{C}_{\mathsf{Abs}}(e, E-1)$ (note that e might be negative). This guarantees that b is indeed the LSB of e_α, as otherwise, e would be close to $(p-1)/2$ and its absolute value cannot fit into $E-1$ bits. Knowing the validity of b, e is in fact in $[-2^{E-2} - M/2, 2^{E-2}]$. Next, we compute the mantissa's square root. To this end, the prover feeds the hint $n \coloneqq H_{\mathsf{Sqrt}}(m_\alpha \ll (M+4+b)) = \sqrt{m_\alpha \ll (M+4+b)}$ to circuit, and the circuit checks the predicate $P_{\mathsf{Sqrt}}(m_\alpha \ll (M+4+b), n)$ by enforcing $n^2 \le (m_\alpha \ll (M+4+b)) < (n+1)^2$ using two range checks. This guarantees that n is (the integer part of) the shifted mantissa's square root. We shift m_α to the left before computing the square root for two reasons: *(i)* when e_α is odd, we decrease it by 1, and thus the mantissa should be doubled when $b = 1$, or equivalently, $m_\alpha \ll b$, and *(ii)* the shift $M+4$ scales m_α to achieve the desired precision. Otherwise, the result n would only have approx. $M/2$ bits of precision. Successively, we obtain the intermediate mantissa $m \coloneqq n$. Recall that the standard requires the intermediate result to have infinite precision, but m is not the exact square root. Thus, we apply the technique introduced in Appendix B: we further compute $r \coloneqq (m_\alpha \ll (M+4+b)) - n^2$, which helps compute the sticky bit in rounding without storing the precise square root. The result is abnormal if α is abnormal or negative (-0 is not included, as $\sqrt{-0} = -0$). Hence, we set $a \coloneqq a_\alpha \vee (s_\alpha \wedge \neg \mathcal{C}_{\mathsf{IsEq}}(m_\alpha, 0))$.

Normalize intermediate mantissa. Now, a non-zero m's upper bound is $\sqrt{(2^{M+1} - 1) \ll (M+5)} < 2^{M+3}$, and its lower bound is $\sqrt{2^M \ll (M+4)} = 2^{M+2}$. Hence, the MSB of m is always 1 and normalization is unnecessary.

Round intermediate mantissa (line 11). The mantissa m of length $N = M+3$ is then rounded as in Section 3.2, with $\Delta e = K = 0$, obtaining e' and m'. Δe is fixed to 0 because the intermediate exponent e of a non-zero result is always greater than $-2^{E-2} - M/2 > -2^{E-1} + 2$, hence we don't need to handle the subnormal case. In addition, the equality between r and 0 is used to determine the sticky bit, thus we set the in-circuit parameter $aux \coloneqq \mathcal{C}_{\mathsf{IsEq}}(r, 0)$.

We omit Edge Cases (lines 12-14) due to space limits.

$\alpha < \beta$
1: $e_ge \coloneqq \mathcal{C}_{\mathsf{GEZ}}(e_\alpha - e_\beta, E+1); m_ge \coloneqq \mathcal{C}_{\mathsf{GEZ}}(m_\alpha - m_\beta, M+1)$
2: $s_lt \coloneqq (\mathcal{C}_{\mathsf{IsEq}}(m_\alpha, 0) \wedge \mathcal{C}_{\mathsf{IsEq}}(m_\beta, 0)) \ ? \ 0 : e_\alpha$
3: $e_lt \coloneqq s_\alpha \ ? \ e_ge : \neg e_ge$
4: $m_lt \coloneqq \mathcal{C}_{\mathsf{IsEq}}(m_\alpha, m_\beta) \ ? \ 0 : (s_\alpha \ ? \ m_ge : \neg m_ge)$
5: $b \coloneqq \mathcal{C}_{\mathsf{IsEq}}(s_\alpha, s_\beta) \ ? \ (\mathcal{C}_{\mathsf{IsEq}}(e_\alpha, e_\beta) \ ? \ m_lt : e_lt) : s_lt$
6: $b' \coloneqq ((a_\alpha \wedge \mathcal{C}_{\mathsf{IsEq}}(m_\alpha, 0)) \vee (a_\beta \wedge \mathcal{C}_{\mathsf{IsEq}}(m_\beta, 0))) \ ? \ 0 : b$
7: **return** b'

Figure 7: Circuit for floating-point comparison

3.6. Comparison

Finally, we discuss the comparison between two IEEE 754 floating-point values $\alpha = (s_\alpha, e_\alpha, m_\alpha, a_\alpha)$ and $\beta = (s_\beta, e_\beta, m_\beta, a_\beta)$ in-circuit. For equality, we support two types of checks: the strict comparison and the fuzzy comparison. The former enforces that α and β are strictly equal by checking if all their components are equal, while the latter asserts that the difference $\alpha - \beta$ is less than a threshold.

Now we introduce the inequality check by using the less than operation as an example. Other comparators like $\le, >, \ge$ can be constructed analogously.

First, we check if α or β is NaN, in which case we return 0. Second, we compare the signs s_α and s_β. If $s_\alpha \neq s_\beta$, then the result is 0 if $\alpha = -0$ and $\beta = +0$ ($-0 = +0$), is 1 if s_α is true but $\alpha \neq -0$ (a negative value is always smaller than a positive one), and 0 otherwise. Now, $s_\alpha = s_\beta$. For the exponent, if $e_\alpha \neq e_\beta$, then the result equals $e_\alpha < e_\beta$ for positive α, β and $e_\alpha > e_\beta$ for negative ones. Otherwise, α and β have the same sign and exponent. We return $m_\alpha < m_\beta$ for positive α, β and $m_\alpha > m_\beta$ for negative ones. Utilizing $\mathcal{C}_{\mathsf{IsEq}}$ and $\mathcal{C}_{\mathsf{GEZ}}$, we translate the above process to in-circuit constraints in Figure 7.

4. Zero Knowledge Location Privacy

In this section, we discuss the technical challenges when evaluating whether a location (θ, ϕ) is in a hexagonal tile $\mathbf{h}$. We provide a simplified description of the H3 protocol [17] that transforms spherical coordinates to hexagonal indices in Figure 8. In the following, we first describe how the baseline algorithm in the H3 hexagonal spatial indexing system transforms (θ, ϕ) into (i, j, k) coordinates, which uniquely identify a hexagonal tile $\mathbf{h}$ in the hexagonal grid. Successively, we discuss how to emulate the transformation in SNARK circuits using all the floating-point circuits in § 3, highlight the difficulties in circuit construction, and introduce optimizations that make the ZKLP paradigm practical.
Transforming Spherical Coordinates to Coordinates in a Discrete Hexagon Planar Grid Systems. To utilize hexagonal hierarchical geospatial indexing, spherical coordinates (i.e., (θ, ϕ)) need to be converted to relative coordinates of the hexagon in the grid system of the geospatial index (i.e., the (i, j, k) coordinates). The H3 coordinate system deterministically maps (i, j, k) coordinates to H3 indices, and the (i, j, k) coordinates, in combination with the resolution res, are sufficient to determine a unique hexagon based on

$\Pi_{\text{Base}}(\theta, \phi, \text{res})$

```
1: (θ_rad, φ_rad) ← ToRadians(θ, φ)                         ▷ Transform to Radians
2: (x_u, y_u, z_u) ← ToCartesian3D(θ_rad, φ_rad)            ▷ Transform to Cartesian
3: d² = (x_F − x_u)² + (y_F − y_u)² + (z_F − z_u)²          ▷ Closest Face
4: r ← CalculateRadialDist(d², res)                         ▷ Calculate the radial distance
5: σ ← CalculateAngle(θ_F, φ_F, θ, φ)                       ▷ Calculate angle to closest Face
6: (x, y) ← ToCartesian2D(r, σ)                             ▷ Calculate 2D Cartestian
7: (I, J, K) = TransformToIJK(x, y)                         ▷ Transform to "I,J,K"
8: return (I, J, K)
```

Figure 8: Baseline Protocol for deriving (i, j, k) from (θ, ϕ).

$\mathcal{C}_{\mathsf{ZKLP}}(\theta_{\text{rad}}, \phi_{\text{rad}}\ ; \text{res}, I, J, K)$

```
1: Receive H_ZKLP(i) = [α_i, β_i, γ_i, δ_i], i ∈ {θ_rad, φ_rad}
2: γ_i² + d_i² = 1 ; δ_i · a_i = γ_i ; 2γ_i · d_i = β_i, i ∈ {θ_rad, φ_rad}
3: r = √(1 − b²_θrad)   ;   z = b_θrad
4: x = √(1 − b²_φrad) · r ;   y = b_φrad · r
5: for i ∈ {0, ..., 19} do
6:     d_i² = (x_Fi − x)² + (y_Fi − y)² + (z_Fi − z)²
7:     d² = (d_i² <? d²) ? d_i² : d²
8: r ← C_rad(d², res)
9: sin_i = β_i ; cos_i = δ_i² − γ_i² for i ∈ {θ_rad, φ_rad}
10: (x, y) ← C_Hex2D(r, res, sin_θrad, cos_θrad, sin_φrad, cos_φrad, F_i)
11: (i, j, k) ← C_IJK(x, y)
12: C_IsEq(i, I) ; C_IsEq(j, J) ; C_IsEq(k, K)
```

Figure 9: Optimized Circuit for computing ZKLP.

the latitude and longitude. In the following, we therefore solely describe the transformation of latitude and longitude to (i, j, k) coordinates in the hexagonal planar grid system of H3. The transformation relies on two logical steps:

(1) Transforming spherical coordinates to Cartesian coordinates: The first step transforms a point from spherical coordinates to Cartesian coordinates in the 2D plane of an icosahedral face, specifically for the hexagonal grid system used in the H3 geospatial indexing system. Given geographic coordinates (θ, ϕ), the distance from a given point on the sphere to the center of the closest face of the icosahedron is computed by evaluating the squared Euclidean distance $d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2$. To do so, (θ, ϕ) are converted to 3D Cartesian coordinates:

$$\begin{aligned} z &= \sin(\theta) \qquad & x &= \cos(\phi) \cdot a \\ a &= \cos(\theta) \qquad & y &= \sin(\phi) \cdot a \end{aligned} \tag{1}$$

To determine the closest icosahedral face to (θ, ϕ), the distance to each face is calculated individually. Given the squared Euclidean distance between two points on a sphere, the algorithm now aims to find the angular distance r between the two points when projected onto a unit sphere. The angular distance between two points on a sphere can be calculated using the spherical law of cosines and relates the sine of half the angular distance to the squared Euclidean distance between the points: $\sin^2\left(\frac{r}{2}\right) = \frac{d^2}{4}$, which can be transformed as $\cos(r) = 1 - 2\sin^2\left(\frac{r}{2}\right) = 1 - 2 \cdot \frac{d^2}{4} = 1 - \frac{d^2}{2}$. This yields the angular distance as $r = \arccos(1 - \frac{d^2}{2})$.

The algorithm performs a gnomonic projection of this angle by taking its tangent $\tan(r)$. The tangent function is used here to convert the angular distance r into a linear distance for the 2D plane. After the gnomonic scaling, r can be thought of as analogous to the radial distance in 2D polar coordinates. The radial distance is scaled according to the scaling factor from hexagonal grid unit length at resolution 0 to gnomonic unit length c_G given the desired H3 resolution res, such that $r = \frac{r}{c_G} \cdot \sqrt{7}^{\text{res}}$. Once the radial distance r is calculated, the counterclockwise angle σ between a reference direction on a given face of an icosahedron (for Uber H3 the i-axis of the Class II orientation [40]) and the direction from the center of that face to a point on the globe is determined. The radial distance r and the angle θ together describe the polar coordinates of the (θ, ϕ) relative to the icosahedron face. The angle σ is calculated as the difference in azimuth (a type of angular measurement in a spherical coordinate system) between a reference axis on the icosahedron face and the azimuth from the center of that face to the given point. The azimuth values are normalized to be between 0 and $2 \cdot \pi$, such that $\sigma = \text{norm}\left(\zeta_{F_i} - \text{norm}\left(\zeta(F_{\text{center}}, (\theta, \phi))\right)\right)$. The azimuth $\zeta(p1, p2)$ from point p1 to point p2, where θ_1, ϕ_1 are the latitude and longitude of p1, and θ_2, ϕ_2 are the latitude and longitude of p2, is calculated as $\zeta(p1, p2) = \arctan(\frac{a}{b})$, with $a = \cos(\theta_2)\sin(\Delta)$, $b = \cos(\theta_1)\sin(\theta_2) - \sin(\theta_1)\cos(\theta_2)\cos(\Delta)$ and $\Delta = \phi_2 - \phi_1$.

In the H3 system, Class II and Class III orientations [13] alternate with each resolution. To adjust for the alternate orientation of hexagons at different resolutions, a constant rotation angle $\arcsin(\sqrt{\frac{3}{28}})$ is subtracted from σ if the chosen resolution is odd. The remaining transformation transforms the coordinates (r, σ) to Cartesian coordinates (x, y) in the two-dimensional plane by computing $x = r \cdot \cos(\sigma)$ and $y = r \cdot \sin(\sigma)$. The resulting coordinates (x, y) are the two-dimensional coordinates of the chosen hexagon relative to the face center of the closest icosahedron.

(2) Transforming (x, y) coordinates to (i, j, k) coordinates: The second step in obtaining coordinates in the discrete hexagonal planar grid system translates two-dimensional Cartesian coordinates (x, y) to three-dimensional coordinates (i, j, k), uniquely identifying a hexagon at a given resolution. It is natural for the grid system to have three coordinate axis, spaced 120 degrees apart from each other, due to the structure of the underlying hexagons. The three-axis system allows for unique addressing without ambiguities [27].

The algorithm initially proceeds with quantization, setting k to 0 and operating on absolute values of (x, y). If the value of (x, y) is not equivalent to the center of the hexagon, the continuous variables are rounded to the nearest hexagon center. To adjust for negative Cartesian coordinates, the Cartesian coordinates are folded across the axes to map them onto the hexagonal grid accordingly in (i, j, k) coordinates. Finally, the computed (i, j, k) coordinates are normalized to ensure that coordinates are as small as possible and non-negative. Normalizing the result is essential in ensuring that each hexagon in the grid has a unique address.

$\mathcal{C}_{\mathsf{rad}}(d^2, \text{res})$

1: $r = \sqrt{\frac{-(d^2-4)\cdot d^2}{(2-d^2)}}$; $r = \frac{r}{c_G}$
2: $\gamma \coloneqq 1.0$; $c_\rho \coloneqq \sqrt{7}$
3: **for** $i \in \{1, \ldots, R\}$ **do**
4: $\quad \gamma \coloneqq (\text{res}[i])$? $\gamma \cdot c_\rho : \gamma$
5: $\quad c_\rho \coloneqq c_\rho^2$
6: **return** $(r \cdot \gamma)$

Figure 10: Optimized Sub-Circuit for computing the radial distance r. R represents the number of bits of res.

$\mathcal{C}_{\mathsf{Hex2D}}(r, \text{res}, \sin_{\theta_{\text{rad}}}, \cos_{\theta_{\text{rad}}}, \sin_{\phi_{\text{rad}}}, \cos_{\phi_{\text{rad}}}, F_i)$

1: $a \coloneqq \sin_{\phi_{\text{rad}}} \cdot \cos_{\phi_{F_i,\text{rad}}}$; $b \coloneqq \cos_{\phi_{\text{rad}}} \cdot \sin_{\phi_{F_i,\text{rad}}}$
2: $c \coloneqq \cos_{\theta_{F_i,\text{rad}}} \cdot \sin_{\theta_{\text{rad}}}$; $d \coloneqq \sin_{\theta_{F_i,\text{rad}}} \cdot \cos_{\theta_{\text{rad}}}$
3: $e \coloneqq \cos_{\phi_{\text{rad}}} \cdot \cos_{\phi_{F_i,\text{rad}}}$; $f \coloneqq \sin_{\phi_{\text{rad}}} \cdot \sin_{\phi_{F_i,\text{rad}}}$
4: $x \coloneqq c - d \cdot (e+g)$; $y \coloneqq \cos_{\theta_{\text{rad}}} \cdot (a-b)$; $z \coloneqq \sqrt{x^2+y^2}$
5: $\sin_{\zeta_{F_i}} \coloneqq (\text{res}[0])$? $(\sin_{\zeta_{F_i}} - \arcsin(\sqrt{\frac{3}{28}}))$: $\sin_{\zeta_{F_i}}$
6: $\cos_{\zeta_{F_i}} \coloneqq (\text{res}[0])$? $(\cos_{\zeta_{F_i}} - \arcsin(\sqrt{\frac{3}{28}}))$: $\cos_{\zeta_{F_i}}$
7: $\sin_\sigma \coloneqq (\sin_{\zeta_{F_i}} \cdot \frac{x}{z}) - (\cos_{\zeta_{F_i}} \cdot \frac{y}{z})$
8: $\cos_\sigma \coloneqq (\cos_{\zeta_{F_i}} \cdot \frac{x}{z}) - (\sin_{\zeta_{F_i}} \cdot \frac{y}{z})$
9: **return** $(r \cdot \sin_\sigma, r \cdot \cos \sigma)$

Figure 11: Optimized Sub-Circuit for computing two-dimensional cartesian coordinates. Variables related to the face center F_i are constant floating-point numbers.

Representing the Transformation as Constraints. SNARKs work by encoding the computation in an arithmetic circuit over a finite field. In contrast, the transformation of geographic coordinates works over real numbers, represented in traditional programs as floating-point values. We address this issue by utilizing the circuits for primitive floating-point operations described in Section 3.

We still face the problem that our primitive operations in Section 3 do not aim to provide precise math functions $(\sin, \cos, \tan, ...)$ as the IEEE standard does not specify the precision of math libraries. Even more so, a naive implementation would require approximation of math functions by the standard three-step recipe, as utilized in standard libraries — range reduction, polynomial approximation, and output compensation. Emulating polynomial approximations, by computing in-circuit Taylor Series, or applying the Remez algorithm [41], would lead to expensive increase in constraints due to high degree polynomials. Further, *precise* approximation is non-trivial, and standard techniques are inefficient without significant in-circuit optimization. Efficient algorithms for emulating accurate trigonometric functions are known for Two-Party-Computation [25]. However, we are not aware of any optimizations that lead to accurate and efficient in-circuit trigonometric approximations for SNARKs. We describe optimizations that fully eliminate trigonometric functions in our circuits as follows.

Avoiding Trigonometric Operations. Recall that transforming spherical coordinates to Cartesian coordinates demands *(i)* calculating the radial distance r of P_u to the closest icosahedral face (Step 4 in Figure 8), *(ii)* calculating the angle σ of P_u to the closest icosahedral face (Step 5 in Figure 8) and *(iii)* converting (r, σ) to Cartesian coordinates (x, y) in the 2D plane by computing $x = r \cdot \cos(\sigma)$ and $y = r \cdot \sin(\sigma)$ (Step 6 in Figure 8). We observe that we can avoid trigonometric operations altogether in the above steps by leveraging trigonometric identities.

To avoid evaluating trigonometric operations for computing the radial distance r, we substitute $r = \arccos(1 - \frac{d^2}{2})$ into $\tan(r)$. By the Pythagorean identity, we express $\tan(r) = \tan(\arccos(1 - \frac{d^2}{2}))$.

$$r = \tan(r) = \sqrt{\frac{(-(d^2-4)\cdot d^2)}{(2-d^2)}} \tag{2}$$

To minimize the number of constraints, we scale to the desired H3 resolution by applying the square and multiply algorithm for bitwise exponentiation instead of naive exponentiation (cf. Figure 10).

Similarly, we simplify computing (x, y) with the angle σ. Recall that $\sigma = \text{norm}\left(\zeta_{F_i} - \text{norm}\left(\zeta(F_{\text{center}}, (\theta, \phi))\right)\right)$ where $\zeta(p1, p2) = \arctan(\frac{a}{b})$. By substituting the above values in the equations for calculating (x, y) and leveraging trigonometric identities, we obtain:

$$x = r \cdot \left(\cos(\theta_{F_i})\frac{b}{\sqrt{a^2+b^2}} + \sin(\theta_{F_i})\frac{a}{\sqrt{a^2+b^2}}\right)$$
$$y = r \cdot \left(\sin(\zeta_{F_i})\frac{b}{\sqrt{a^2+b^2}} - \cos(\zeta_{F_i})\frac{a}{\sqrt{a^2+b^2}}\right)$$

Note, that the trigonometric identities used in the simplification are mathematically sound regardless of normalization. As the center point of each icosahedron is fixed and known, the remaining $\sin$ and $\cos$ terms can be precomputed. This representation is significantly less costly, as we already derived an optimized gadget for computing the square root of a floating-point variable in Section 3.5.

Eliminating Trigonometric Operations with Hints. It remains the elimination of trigonometric operations by avoiding the initial calculation of the Cartesian coordinates (x, y, z) from the user-supplied spherical coordinates (θ, ϕ) (cf. Equation 1). We observe that we can mindfully construct hints, such that the in-circuit computation reduces to *(i)* evaluating the hint predicate and *(ii)* calculating (x, y, z) without using trigonometric functions. As such, the hint $H_{\mathsf{ZKLP}}(\theta) = [\alpha_\theta, \beta_\theta, \gamma_\theta, \delta_\theta]$ is computed as $\gamma_\theta = \sin\left(\frac{\theta}{2}\right)$, $\delta_\theta = \cos\left(\frac{\theta}{2}\right)$, $\beta_\theta = 2 \cdot \gamma_\theta \cdot \delta_\theta$ and $\alpha_\theta = \tan\left(\frac{\theta}{2}\right)$.

Soundness holds as P_{ZKLP} evaluates that *(i)* $\gamma_\theta^2 + \delta_\theta^2$ equals 1, and thereby fulfills the fundamental trigonometric identity, *(ii)* $\delta_\theta \cdot \alpha_\theta$ equals γ_θ, which checks if the same angle is used, and *(iii)* $2 \cdot \gamma_\theta \cdot \delta_\theta$ equals β_θ, confirming that β_θ is correctly related to γ_θ and δ_θ. Afterwards, the x,y,z coordinates can simply be derived as $z = \beta_\theta$, $r = \sqrt{1 - \beta_\theta^2}$, $x = \sqrt{1 - \beta_\phi^2} \cdot r$ and $y = b_\phi \cdot r$. As a result, trigonometric operations are eliminated from $\mathcal{C}_{\mathsf{ZKLP}}$. Note that $\sin(i) = \beta_i$ and $\cos(i) = \delta_i^2 - \gamma_i^2$ holds for $i \in \{\theta, \phi\}$.

Transforming (x, y) to (i, j, k) coordinates. The transfor-

mation is conducted by first transforming (x, y) to (i, j, k) coordinates and successively normalizing (i, j, k) coordinates to adjust for negative coordinates. We provide a detailed description of the sub-circuits for obtaining and normalizing (i, j, k) coordinates in Figure 18 and 19 in the appendix. They directly benefit from our floating-point implementation, due to many floating-point comparisons.

5. Empirical Evaluation

Our empirical evaluation addresses three questions: *(i)* What is the performance and accuracy of our floating-point implementation?, *(ii)* How effective are our optimizations for the ZKLP paradigm?, and *(iii)* How tolerable is the cost of ZKLP in real-world use?

Implementation. We implement the floating-point primitive operations (cf. §3) as a reusable library in gnark [36]. We provide a full implementation of the optimized circuits for ZKLP (cf. §4) for FP32 and FP64 values. In addition, we implement the baseline protocol, without the optimizations as described in §4, over fixed-point arithmetic. Due to the agnostic nature of gnark, our implementation supports Groth16 and Plonk as the SNARK. We instantiate the lookup argument as LogUp [35], which is used in gnark for range checks. Our implementation and measurement data are open-sourced [42] for reproducibility.

Test Suite. To ensure compliance with IEEE 754 [16], we create a set of test values with the Berkeley TestFloat library [26], which generates test cases to ensure that an implementation conforms to the IEEE Standard for Floating-Point Arithmetic. Specifically, 46464 test cases for each binary operation (e.g., Add), and 600 test cases for the unary operation Sqrt are created. Our implementation passes all these test cases, including inputs with abnormal values.

To evaluate the ZKLP circuit, we generate a series of geospatial points within hexagonal cells at various resolution levels, using Uber H3 implemented in C [17]. For each resolution res $\in [0, 15]$, we test 16 distances from the center to the point. The i-th distance is $(1 - 2^{-i})d$, where d is the center-to-boundary distance. We further randomly sample 100 points at each distance. Thus, we have a sparser distribution of points closer to the center and denser as it approaches the boundary (cf. Figure 1). In addition, because the test cases generated by Uber H3 contain floating-point values, when evaluating the fixed-point baseline, we convert the floating-point values to fixed-point by multiplying by a scalar and rounding to the nearest integer.

Experimental Setup. When evaluating the runtime and memory consumption of a circuit, we execute all tests on an `m6i.xlarge` AWS instance with 4 vCPUs and 16 GB of RAM. The number of constraints is independent of the execution architecture. For running time we report all values as the mean of 20 executions with multi-threading enabled.

5.1. Microbenchmarks and Comparison

We evaluate the cost of floating-point circuits (Section 3) and compare our implementation with existing works.

TABLE 1: Number of R1CS constraints for in-circuit floating-point primitive operations, with $|\mathcal{T}_{\mathsf{RC}}| = 2^8, |\mathcal{T}_{\mathsf{Pow2}}| = 1 + E + M$

FP32 Operation		**Init**	**Add/Sub**	**Mul**	**Div**	**Sqrt**	**Cmp**
# Native Constraints		13	42	31	38	23	26
# Lookup Constraints	(i)	17	43	33	38	22	7
	(ii, iii)	291					
FP64 Operation		**Init**	**Add/Sub**	**Mul**	**Div**	**Sqrt**	**Cmp**
# Native Constraints		13	42	31	38	23	26
# Lookup Constraints	(i)	32	71	57	60	38	11
	(ii, iii)	323					

Number of Constraints. Table 1 presents the number of R1CS constraints for in-circuit floating-point primitive operations. As discussed in Appendix A, we utilize two lookup tables $\mathcal{T}_{\mathsf{RC}}$ and $\mathcal{T}_{\mathsf{Pow2}}$, where the former is for range check, and the latter is for the computation of 2^d. In Table 1, the size of $\mathcal{T}_{\mathsf{RC}}$ is fixed at 2^8, and the size of $\mathcal{T}_{\mathsf{Pow2}}$ is 32 for FP32 and 64 for FP64. The constraints for each operation consist of two parts: native constraints (supported by the constraint system) and lookup constraints (for lookup tables). The inclusion of lookup constraints is necessary due to gnark's implementation of LogUp [35]. Recall that gnark checks LogUp's identity for set inclusion $\sum_{i=1}^{|\mathbf{f}|} \frac{1}{X - f_i} = \sum_{j=1}^{|\mathbf{t}|} \frac{o_j}{X - t_j}$ in arithmetic circuit, resulting in a single SNARK proof for both the original relation and the validity of set inclusion. To eliminate lookup constraints, one can instantiate the lookup argument as standalone protocol and connect it to SNARK in a commit-and-prove fashion, at the cost of larger proofs.

Specifically, the in-circuit verification of LogUp's identity involves three steps: *(i)* compute the LHS $\sum_{i=1}^{|\mathbf{f}|} \frac{1}{X - f_i}$, *(ii)* compute the RHS $\sum_{j=1}^{|\mathbf{t}|} \frac{o_j}{X - t_j}$, and *(iii)* check the equality between the LHS and the RHS. As highlighted in Table 1, the costs of *(ii)* and *(iii)* are operation-agnostic; they only depend on the sizes of lookup tables $\mathcal{T}_{\mathsf{RC}}$ and $\mathcal{T}_{\mathsf{Pow2}}$. Hence, costs remain unchanged as the number of queries increases and they can be amortized across multiple operations.

Figure 12 shows the amortized cost of primitive operations, using multiplication as an example. With a larger $\mathcal{T}_{RC}$, the cost of step *(ii)* increases, while step *(i)* requires fewer constraints. This is particularly advantageous when proving the execution of many operations, as the one-time cost of steps *(ii)* and *(iii)* becomes negligible.

Comparison With Other Works. Naively converting FP32 operations compliant to IEEE 754 requires 2456 and 8854 boolean gates for addition and multiplication [22]. FP64 addition and multiplication require 15637 and 44899 boolean gates respectively [21]. Garg *et. al* [23] provide the state-of-the-art succinct ZKP for floating-point operations, which requires 108 non-zero entries in the R1CS instance for FP32 addition and 25 for FP32 multiplication in a circuit over BN254. Due to the inconsistent metrics and our requirement of amortization, we are unable to present a fair comparison.

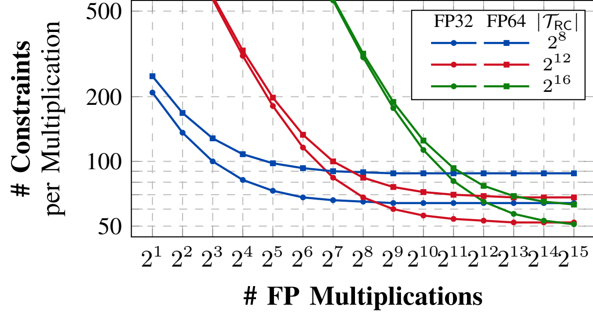


Figure 12: Number of Constraints per Multiplication vs. Number of Multiplications, for Groth16. $|\mathcal{T}_{RC}|$ is the size of the lookup table for the range check (cf. Section 3).

5.2. Zero-Knowledge Location Proofs

We assess the efficiency of end-to-end ZKLP circuits (cf. Section 4) and highlight their relevance in privacy-preserving peer-to-peer proximity testing as a case study. Our analysis includes a baseline (Π_{Base}, cf. Figure 8), implemented via unoptimized fixed-point circuits that support two fixed-point data types P20 and P40. The scaling factors for P20 and P40 are $10^6 \approx 2^{20}$ and $10^{12} \approx 2^{40}$, i.e., their last 20 and 40 bits are fractional parts. To represent large intermediate results, we do not limit the number of bits used for the integer parts of P20 and P40, as long as they are smaller than the order of $\mathbb{F}_p$. Specifically, for $\sqrt{7}^{\text{res}}$ with res $= 15$, the integer parts must use at least 23 bits, and subsequent operations even require P20 and P40 to have 42 bits and 81 bits in their integer parts. In total, the maximum bit lengths for P20 and P40 are 62 bits and 121 bits, respectively. In the fixed-point circuits, we approximate trigonometric functions within the circuit: $\sin(x)$ is approximated via a Taylor Series expansion, and $\arctan(x)$ is approximated using the Remez algorithm [41]. The mathematical function approximation follows the standard three-step method: *range reduction*, *polynomial approximation*, and *output compensation* [43]. In contrast, we present results for our optimized floating-point circuits, implementing the $\mathcal{C}_{\mathsf{ZKLP}}$ circuit (cf. Figure 9).

Quantitative Baseline Comparison. Table 2 depicts the quantitative comparison of the fixed-point baseline and $\mathcal{C}_{\mathsf{ZKLP}}$ implemented over our floating-point circuits. Our results show, that our optimizations applied for $\mathcal{C}_{\mathsf{ZKLP}}$ are indeed very effective. With Groth16, our single precision floating-point circuit has $15.9\times$ less constraints than the fixed-point baseline, whereas with double precision floating-point, our circuit has $12.2\times$ less constraints than the baseline. Note that the number of constraints in $\mathcal{C}_{\mathsf{ZKLP}}$ remains constant, regardless of the chosen resolution. The proof generation time for both $\mathcal{C}_{\mathsf{ZKLP}}$ over FP32 and FP64 is below 1 s for Groth16. With Plonk, the time to generate a proof is higher, which is expected given Plonk's prover complexity $\mathcal{O}(n \log n)$ given an arithmetic circuit of n gates.

While we evaluate $\mathcal{C}_{\mathsf{ZKLP}}$ on a server, mid-range mobile devices can achieve similar performance. On an Android phone with a Snapdragon 7+ Gen 2 CPU, the prover takes 0.256 s for FP32 and 0.333 s for FP64 with Groth16 as the SNARK, demonstrating the practicality of ZKLP.

TABLE 2: Evaluation of $\mathcal{C}_{\mathsf{ZKLP}}$ over BN254 for the floating-point implementation opposed to the baseline protocol Π_{Base} (cf. Figure 8), implemented with fixed-point arithmetic P20. For Groth16, the SRS size is the size of the prover key.

Circuit	Type	Proof System	# Constraints (x10^3)	Prover Time	Memory (MB)	SRS (MB)
Baseline	P20	Groth16	309.6	1.75 s	517.3	52.4
$\mathcal{C}_{\mathsf{ZKLP}}$	FP32	Groth16	19.5	0.26 s	248.7	4.6
	FP64	Groth16	25.5	0.32 s	246.7	6.2
Baseline	P20	Plonk	570.9	38.42 s	1654.1	33.6
$\mathcal{C}_{\mathsf{ZKLP}}$	FP32	Plonk	55.5	2.19 s	249.1	2.1
	FP64	Plonk	81.3	4.41 s	247.1	4.2

Qualitative Baseline Comparison. We report the success rate of tests for fixed-point and floating-point implementations for emulating the transformation of (θ, ϕ) to (i, j, k) coordinates in a ZKP for resolutions 0 to 15 (Figure 13). We find that in the baseline protocol with P20, errors start to occur frequently at a resolution of res $= 3$, with most tests failing for res ≥ 11. In contrast, errors in the modified protocol become frequent only for res ≥ 7, and significant failures are observed only when res $= 15$. After adjusting the fixed-point implementation to P40, the errors are greatly reduced, though there are still some edge cases where P40 falls short. On the other hand, all tests pass with FP64, as its data format and rounding mode align with the Uber H3 implementation. Overall, we find that both the precision and the rounding mode have a significant impact on the accuracy of the computations, e.g., those involving the constant scaling factor of the resolution ($\sqrt{7}^{\text{res}}$).

P2P Proximity Testing. We utilize the ZKLP paradigm to realize P2P proximity testing. Let $\mathcal{H} = \{\mathbf{h_1}, \mathbf{h_2}, ..., \mathbf{h_n}\}$ be a set of hexagons generated by the H3 system, where each hexagon $\mathbf{h}$ is defined by its boundary vertices. Each vertex v_i of hexagon $\mathbf{h}$ is given in geographical coordinates (latitude θ_i and longitude ϕ_i). Given a user's position $P_u = (\theta_u, \phi_u)$, the proximity to a hexagonal cell $\mathbf{h}_v$ is evaluating the Haversine formula to calculate the great-circle distance between P_u and each vertex of $\mathbf{h}_v$. The Haversine distance δ can be calculated as $c = 2 \cdot \operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$, where $a = \sin^2\left(\frac{\theta_2 - \theta_1}{2}\right) + \cos(\theta_1) \cdot \cos(\theta_2) \cdot \sin^2\left(\frac{\phi_2 - \phi_1}{2}\right)$. Here, θ_1, ϕ_1 and θ_2, ϕ_2 are the latitude and longitude in radians of points P_u and a vertex of $\mathbf{h}_v$, respectively, and r is the radial length of the earth. The minimum distance Δ_{min} from point P_u to the boundary of hexagon $\mathbf{h}_v$ is calculated as the smallest distance $\Delta_{\text{min}} = \min(\delta_0, \delta_1, ..., \delta_i)$ (cf. Figure 1). Calculating the proximity of P_u to the hexagonal boundary is done in plain and hence efficient when compared to proving the ZKLP circuit. We implement proximity testing in Go, relying on the H3 library in C to determine the boundary points of the hexagon. We find that the computation requires $\approx 581.87\,\mu$s to execute, which is comparable to verifying a Groth16 proof (≈ 1.54 ms). Hence, a verifier can evaluate its proximity to ≈ 470 peers per second.

6. Related Works

Floating-Point Secure Computing. To the best of our knowledge, there is no prior work supporting fully IEEE 754 compliant floating point computations for succinct proof systems. There are several prior works that investigate floating-point computations for secure MPC [19], [20]. Later, [21], [44] build IEEE 754 compliant MPC circuits by compiling from software implementations of IEEE 754 using tools such as CBMC-GC. However, the resulting circuits have at least thousands of gates per operation, making them inefficient in practice. In [25], Rathee *et. al* construct standard compliant functionalities for 2PC with dedicated optimizations. While providing better efficiency, they can only achieve partial compliance with IEEE 754, but subnormal values and NaNs are not considered. Another line of research focuses on proving floating-point computations using ZKPs. Weng *et. al* [22] use the IEEE 754 compliant single-precision boolean circuits from EMP-toolkit [45] with non-succinct proofs. Closest to our work is the work of Garg *et. al* [23], which studies succinct ZKPs for floating-point arithmetic. However, their approach only supports addition and multiplication, whilst not providing a concrete implementation and only theoretical performance estimates. Also, the verifier time in [23] is linear. While they provide a method to achieve sub-linear verification, the best complexity they could achieve is $O(\sqrt{n})$, where n is the circuit size. In comparison, due to Groth16, the verifier time of our construction is constant in n. Further, the relative error model in [23] is not suitable for many applications. For instance, it is observed in [18] that for machine learning, even the minor rounding errors due to non-determinism in GPUs can result in very different predictions. Consequently, an adversary may leverage this fact to generate valid proofs for arithmetic circuits that do not produce the expected results as in IEEE 754 compliant computer hardware.

Location Privacy. There is a long line of work on LPPM via geo-indistinguishability [3], [8], [46], [47]. Narayanan *et. al* [7] introduce location privacy via private proximity testing. Vsedvenka *et. al* [8] introduce interactive protocols for proximity testing over a spherical surface. In a similar setting, their protocols require $> 1\,\mathrm{s}$ and > 10 kB communication. In contrast, our protocol is non-interactive and requires $\approx 0.26\,\mathrm{s}$ execution time (disregarding latency) and communicating a ≈ 200 Byte proof with Groth16. Babel *et. al* [48] show how to evaluate whether a location is in a polygon. However, their approach assumes coordinates in (x, y) form in Euclidean plane. In comparison, our circuit for transforming (x, y) to a hexagonal index ($\mathcal{C}_{\mathsf{IJK}}$) yields $\approx 11.7\times$ less constraints. To the best of our knowledge, ZKLP provides the first paradigm for non-interactive, publicly verifiable and privacy preserving proofs of geolocation.

7. Discussion & Future Work

This paper introduces ZKLP (§ 4) via accurate floating-point SNARKs (§ 3), identifying a novel class of applications of general-purpose succinct ZK proofs. The main challenge was the accurate emulation of floating-point values without increasing constraints or compromising soundness.

In our experiments, we show that our implementation of floating-point arithmetic is efficient and accurate. We show that our instantiation of lookup tables amortizes the number of constraints per primitive operation. We show that the ZKLP paradigm is efficient, allowing users to generate an accurate proof in $0.26\,\mathrm{s}$. We instantiate ZKLP with P2P proximity testing and show that a verifier can verify proximity to up to 470 peers per second. Note, that the verification time and proof size is inherited by Groth16 [49], [50].

ZKLP can be directly applied to scenarios where the location data is already authenticated. For example, in the workflow of C2PA [51], the location where a photo is taken is signed by a C2PA-compatible camera, and thus we can seamlessly integrate ZKLP with C2PA to provide photo authenticity while obfuscating the accurate location, thereby preserving the privacy of the photo's author. Further, we identify three potential solutions on how to obtain authentic location, which we detail in Appendix C.

We expect our results to adapt naturally to other settings. For instance, in machine learning, where parameters are often floating-point numbers [52], our methods can enable efficient, precise training [24] and inference proofs [14]. Finally, we believe that authenticated ZKLP could be a useful building block in applications for Proof-of-Personhood to obtain verifiable location-based Sybil-resistance [53] and leave its exploration for future work.

Acknowledgments

We acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt.". Joint project 6G-life, project identification number: 16KISK002. This work was done while Chengru Zhang was visiting UCL.

References

[1] S. Boukoros, M. Humbert, S. Katzenbeisser, and C. Troncoso, "On (the lack of) location privacy in crowdsourcing applications," in *28th USENIX Security Symposium*, 2019, pp. 1859–1876.

[2] H. Jiang, J. Li, P. Zhao, F. Zeng, Z. Xiao, and A. Iyengar, "Location privacy-preserving mechanisms in location-based services: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–36, 2021.

[3] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: differential privacy for location-based systems," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 901–914.

[4] B. Lee, J. Oh, H. Yu, and J. Kim, "Protecting location privacy using location semantics," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 1289–1297.

[5] A. Khoshgozaran and C. Shahabi, "Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy," in *International symposium on spatial and temporal databases*. Springer, 2007, pp. 239–257.

[6] R. A. Popa, H. Balakrishnan, and A. J. Blumberg, “Vpriv: Protecting privacy in location-based vehicular services,” 2009.

[7] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, D. Boneh *et al.*, “Location privacy via private proximity testing.” in *NDSS*, vol. 11, 2011.

[8] J. Šeděnka and P. Gasti, “Privacy-preserving distance computation and proximity testing on earth, done right,” in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 99–110.

[9] A.-M. Olteanu, K. Huguenin, R. Shokri, M. Humbert, and J.-P. Hubaux, “Quantifying interdependent privacy risks with location data,” *IEEE Transactions on Mobile Computing*, vol. 16, no. 3, pp. 829–842, 2016.

[10] P. Zhang, S. G. Nagarajan, and I. Nevat, “Secure location of things (slot): Mitigating localization spoofing attacks in the internet of things,” *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2199–2206, 2017.

[11] M. Yuan, X. Tang, and G. Ou, “Authenticating gnss civilian signals: A survey,” *Satellite Navigation*, vol. 4, no. 1, pp. 1–18, 2023.

[12] A. Heinrich, M. Stute, T. Kornhuber, and M. Hollick, “Who can find my devices? security and privacy of apple’s crowd-sourced bluetooth location tracking system,” *arXiv preprint arXiv:2103.02282*, 2021.

[13] K. Sahr, D. White, and A. J. Kimerling, “Geodesic discrete global grid systems,” *Cartography and Geographic Information Science*, vol. 30, no. 2, pp. 121–134, 2003.

[14] D. Kang, T. Hashimoto, I. Stoica, and Y. Sun, “Scaling up trustless dnn inference with zero-knowledge proofs,” *arXiv preprint arXiv:2210.08674*, 2022.

[15] ——, “Zk-img: Attested images via zero-knowledge proofs to fight disinformation,” *arXiv preprint arXiv:2211.04775*, 2022.

[16] “IEEE standard for floating-point arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[17] Uber, “Uber hexagonal hierarchical spatial index,” https://www.uber.com/en-DE/blog/h3/, 2023.

[18] M. Srivastava, S. Arora, and D. Boneh, “Optimistic verifiable training by controlling hardware nondeterminism,” 2024.

[19] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele, “Secure computation on floating point numbers,” in *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013.

[20] L. Kamm and J. Willemson, “Secure floating point arithmetic and private satellite collision analysis,” *International Journal of Information Security*, vol. 14, no. 6, pp. 531–548, 2015.

[21] D. W. Archer, S. Atapoor, and N. P. Smart, “The cost of ieee arithmetic in secure computation,” in *Progress in Cryptology – LATINCRYPT 2021*, 2021, pp. 431–452.

[22] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, “Mystique: Efficient conversions for {Zero-Knowledge} proofs with applications to machine learning,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 501–518.

[23] S. Garg, A. Jain, Z. Jin, and Y. Zhang, “Succinct zero knowledge for floating point computations,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, p. 1203–1216.

[24] S. Garg, A. Goel, S. Jha, S. Mahloujifar, M. Mahmoody, G.-V. Policharla, and M. Wang, “Experimenting with zero-knowledge proofs of training,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1880–1894.

[25] D. Rathee, A. Bhattacharya, R. Sharma, D. Gupta, N. Chandran, and A. Rastogi, “Secfloat: Accurate floating-point meets secure 2-party computation,” in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 576–595.

[26] J. Hauser, “Berkeley testfloat release 3e,” https://github.com/ucb-bar/berkeley-testfloat-3, 2018.

[27] K. Sahr, “Central place indexing: Hierarchical linear indexing systems for mixed-aperture hexagonal discrete global grid systems,” *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 54, no. 1, pp. 16–29, 2019.

[28] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, “From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again,” in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2012, p. 326–349.

[29] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, “Quadratic span programs and succinct nizks without pcps,” in *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*. Springer, 2013, pp. 626–645.

[30] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, “Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge,” *Cryptology ePrint Archive*, 2019.

[31] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, “Scalable, transparent, and post-quantum secure computational integrity,” *Cryptology ePrint Archive*, 2018.

[32] A. L. Xiong, B. Chen, Z. Zhang, B. Bünz, B. Fisch, F. Krell, and P. Camacho, “{VeriZexe}: Decentralized private computation with universal setup,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4445–4462.

[33] J. Bootle, A. Cerulli, J. Groth, S. Jakobsen, and M. Maller, “Arya: Nearly linear-time zero-knowledge proofs for correct program execution,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2018, pp. 595–626.

[34] T. Solberg, “A brief history of lookup arguments,” https://github.com/ingonyama-zk/papers/blob/main/lookups.pdf, 2023.

[35] U. Haböck, “Multivariate lookups based on logarithmic derivatives,” *Cryptology ePrint Archive*, 2022.

[36] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie, “Consensys/gnark: v0.9.0,” Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.5819104

[37] A. Naveh and E. Tromer, “Photoproof: Cryptographic image authentication for any set of permissible transformations,” in *2016 IEEE Symposium on Security and Privacy*. IEEE, 2016, pp. 255–271.

[38] S. Angel, A. J. Blumberg, E. Ioannidis, and J. Woods, “Efficient representation of numerical optimization problems for SNARKs,” in *31st USENIX Security Symposium*, 2022, pp. 4273–4290.

[39] E. Süli and D. F. Mayers, *An introduction to numerical analysis*. Cambridge university press, 2003.

[40] E. S. Popko and C. J. Kitrick, “Divided spheres,” in *Divided Spheres*. AK Peters/CRC Press, 2021, pp. 1–12.

[41] E. Y. Remez, “Sur la détermination des polynômes d’approximation de degré donnée,” *Comm. Soc. Math. Kharkov*, vol. 10, no. 196, pp. 41–63, 1934.

[42] “Floating point and zklp open-source implementation,” https://anonymous.4open.science/r/zk-Location-8B30/, 2024.

[43] W. J. Cody, *Software Manual for the Elementary Functions (Prentice-Hall series in computational mathematics)*. Prentice-Hall, Inc., 1980.

[44] P. Pullonen and S. Siim, “Combining secret sharing and garbled circuits for efficient private ieee 754 floating-point computations,” in *Financial Cryptography and Data Security: FC 2015 International Workshops, BITCOIN, WAHC, and Wearable*. Springer, 2015, pp. 172–183.

[45] X. Wang, A. J. Malozemoff, and J. Katz, “EMP-toolkit: Efficient MultiParty computation toolkit,” https://github.com/emp-toolkit, 2016.

[46] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J.-P. Hubaux, “Quantifying location privacy,” in *2011 IEEE symposium on security and privacy*. IEEE, 2011, pp. 247–262.

[47] R. Shokri, G. Theodorakopoulos, C. Troncoso, J.-P. Hubaux, and J.-Y. Le Boudec, “Protecting location privacy: optimal strategy against localization attacks,” in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 617–627.

[48] M. Babel and J. Sedlmeir, “Bringing data minimization to digital wallets at scale with general-purpose zero-knowledge proofs,” *arXiv preprint arXiv:2301.00823*, 2023.

[49] J. Groth, “On the size of pairing-based non-interactive arguments,” in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2016, pp. 305–326.

[50] J. Ernstberger, S. Chaliasos, G. Kadianakis, S. Steinhorst, P. Jovanovic, A. Gervais, B. Livshits, and M. Orrù, “zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks,” *Cryptology ePrint Archive*, 2023.

[51] “The coalition for content provenance and authenticity (c2pa),” https://c2pa.org/.

[52] T. Yeh, M. Sterner, Z. Lai, B. Chuang, and A. Ihler, “Be like water: Adaptive floating point for machine learning,” in *International Conference on Machine Learning*. PMLR, 2022, pp. 25 490–25 500.

[53] M. Borge, E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, and B. Ford, “Proof-of-personhood: Redemocratizing permissionless cryptocurrencies,” in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2017, pp. 23–26.

[54] G. Beck, H. Eldridge, M. Green, N. Heninger, and A. Jain, “Abuse-resistant location tracking: Balancing privacy and safety in the offline finding ecosystem,” *Cryptology ePrint Archive*, 2023.

[55] M. Orrù, G. Kadianakis, M. Maller, and G. Zaverucha, “Beyond the circuit: How to minimize foreign arithmetic in zkp circuits,” *Cryptology ePrint Archive*, 2024.

[56] A. Galan, I. Fernandez-Hernandez, L. Cucchi, and G. Seco-Granados, “Osnmalib: An open python library for galileo osnma,” in *10th Workshop on Satellite Navigation Technology*. IEEE, 2022, pp. 1–12.

[57] X. Xie, K. Yang, X. Wang, and Y. Yu, “Lightweight authentication of web data via garble-then-prove,” *Cryptology ePrint Archive*, 2023.

[58] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, “Deco: Liberating web data using decentralized oracles for tls,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1919–1938.

[59] J. Lauinger, J. Ernstberger, A. Finkenzeller, and S. Steinhorst, “Janus: Fast privacy-preserving data provenance for tls 1.3,” *Cryptology ePrint Archive*, 2023.

[60] J. Ernstberger, J. Lauinger, Y. Wu, A. Gervais, and S. Steinhorst, “Origo: Proving provenance of sensitive data with constant communication,” *Cryptology ePrint Archive*, 2024.

Appendix A. Circuits for Integer Operations

Arithmetic circuits only natively support operations in the prime field $\mathbb{F}_p$, but it is non-trivial to perform in-circuit integer operations in an efficient and sound way. Below we introduce circuits that checks the range, extracts sign and absolute value, computes maximum and minimum values, and performs left and right shifts for integers.

Range Check. Bit decomposition is a standard technique for ensuring a variable $v \in \mathbb{F}_p$ is an L-bit string, i.e., $v \in [0, 2^L - 1]$. The prover first provides the decomposition as a hint to the circuit by computing $H_{\mathsf{Dec}}(v) = \{v_0, \dots, v_{L-1}\}$. The predicate P_{Dec} for verifying the bit decomposition of v into $\{v_0, \dots, v_{L-1}\}$ asserts that all variables are boolean by checking $v_i(1 - v_i) = 0$ for all $i \in [0, L-1]$, and that v is indeed composed of $\{v_0, \dots, v_{L-1}\}$ by asserting $\sum_{i=0}^{L-1} 2^i v_i = v$. P_{Dec} returns 1 if these checks pass and 0 otherwise. Note, that L should satisfy $2^L < p$ to ensure $\sum 2^i v_i$ does not overflow. For a and b known to be L-bit strings, this method can be extended to ensure $v \in [a, b]$ by decomposing $v - a$ and $b - v$ separately into L bits.

However, bit decomposition requires $L + 1$ constraints, which is not optimal. We can improve the circuit efficiency by leveraging lookup argument. We build a lookup table $\mathcal{T}_{\mathsf{RC}}$ with entries $\{0, \dots, 2^T - 1\}$. On inputs v, L, the circuit $\mathcal{C}_{\mathsf{RC}}$ now requires the prover to compute the hint $H_{\mathsf{Dec'}}(v)$ by decomposing v into T-bit strings $v'_0, \dots, v'_{L/T-1}$. Then the circuit checks the predicate $P_{\mathsf{Dec'}}(v, \{v'_0, \dots, v'_{L/T-1}\})$ by appending the set $\{v'_0, \dots, v'_{L/T-1}\}$ to the vector of queries $\mathbf{t}_{\mathsf{RC}}$ and enforcing $v = \sum_{i=0}^{L/T-1} 2^{iT} v'_i$.

The $\mathcal{C}_{\mathsf{RC}}$ circuits based on both approaches for checking $v \in [0, 2^L - 1]$ are described in Figure 14.

Sign and Absolute Value. Obtaining the sign and the L-bit absolute value of a variable v presents a more complex challenge. Intuitively, a number is positive if it is greater than 0, and is negative otherwise. However, as the field $\mathbb{F}_p$ is not *ordered*, we cannot *compare* between its elements. To address this, we manually define elements in the set $\{1, 2, \dots, (p-1)/2\}$ as positive, and those in the set $\{(p+1)/2, \dots, p-2, p-1\}$ as negative. Now, as long as $2^L < (p-1)/2$, we can extract the sign and the absolute value of v as depicted in $\mathcal{C}_{\mathsf{Abs}}$ in Figure 15. The prover determines if v is positive by checking which set it belongs to, and provides $H_{\mathsf{GEZ}} = s$ as a hint to the circuit. The gadget enforces that s is boolean, and computes v's absolute value abs, which is v if s is 1, and is $-v$ otherwise. Finally, the gadget enforces that abs has at most L bits and returns abs and s. Soundness holds because if an adversary feeds the incorrect s to the circuit, then abs's value belongs to the negative set and is hence greater than $(p-1)/2$, but $\mathcal{C}_{\mathsf{RC}}$ is later used to guarantee that $abs < 2^L < (p-1)/2$.

Maximum and Minimum. Given $\mathcal{C}_{\mathsf{Abs}}$, it is straightforward to build circuits that find the maximum and minimum values between x and y whose difference has L bits (cf. Figure 15), which is done by calling $\mathcal{C}_{\mathsf{Abs}}$ on $x - y$ and select x or y based on the sign of the difference. $\mathcal{C}_{\mathsf{Max}}$ and $\mathcal{C}_{\mathsf{Min}}$ forward $x - y$ and L to $\mathcal{C}_{\mathsf{Abs}}$, where $\mathcal{C}_{\mathsf{Abs}}$ returns the sign bit s only.

Shifting. Figure 16 summarizes the circuits for shifting left $\ll$ and right $\gg$. First, we compute powers of 2 in-circuit. For a constant exponent D, 2^D is also a constant and does not involve any constraints. For 2^d with a variable exponent $d \in [0, K]$ where $2^K < p$, one approach is to leverage the square-and-multiply algorithm. That is, we decompose d into K bits $d_0, \dots, d_{K-1}$ and select the term 2^{2^i} or 1 based on the value of d_i. The product of these terms is the result 2^d. This method costs $O(K)$ constraints. For example, in R1CS, we need $K + 1$ constraints for bit decomposition and $K - 1$ constraints for multiplying K terms, resulting in $2K$ constraints in total. This is not ideal, especially for our

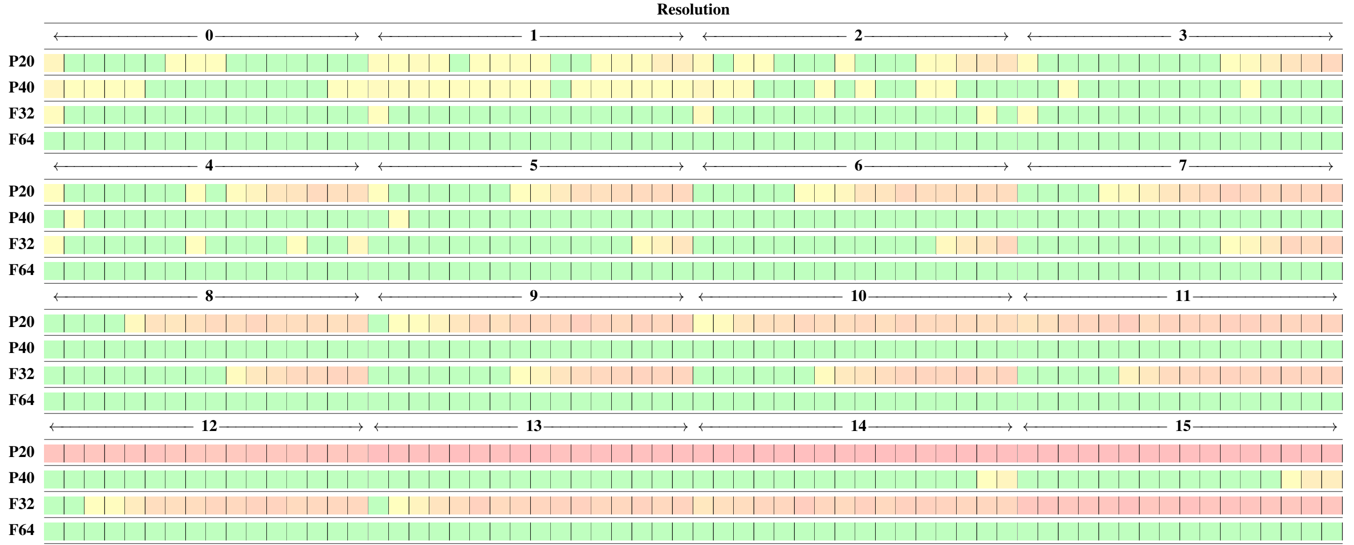
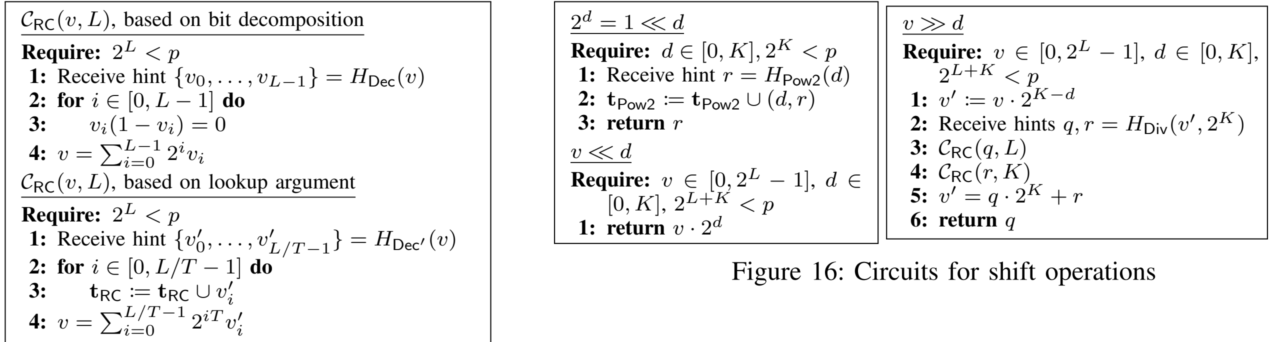


Figure 13: Testing proof generation for ZKLP with resolutions 0 to 15 for fixed-point (P20, P40), single precision (FP32) and double precision (FP64) floating-point values. For a given resolution, we test 16 different distances from the test case to the boundary, with 100 randomly sampled test cases at each distance. All tests are for Groth16 over BN254. Green = pass all tests, Yellow and Red = fail some tests. A cell with deeper red indicates more failures.



$\mathcal{C}_{\mathsf{RC}}(v, L)$, based on bit decomposition

Require: $2^L < p$
1: Receive hint $\{v_0, \ldots, v_{L-1}\} = H_{\mathsf{Dec}}(v)$
2: **for** $i \in [0, L-1]$ **do**
3: $\quad v_i(1 - v_i) = 0$
4: $v = \sum_{i=0}^{L-1} 2^i v_i$

$\mathcal{C}_{\mathsf{RC}}(v, L)$, based on lookup argument

Require: $2^L < p$
1: Receive hint $\{v'_0, \ldots, v'_{L/T-1}\} = H_{\mathsf{Dec}'}(v)$
2: **for** $i \in [0, L/T-1]$ **do**
3: $\quad \mathbf{t}_{\mathsf{RC}} := \mathbf{t}_{\mathsf{RC}} \cup v'_i$
4: $v = \sum_{i=0}^{L/T-1} 2^{iT} v'_i$

Figure 14: Circuit for checking $v \in [0, 2^L - 1]$.

$2^d = 1 \ll d$

Require: $d \in [0, K], 2^K < p$
1: Receive hint $r = H_{\mathsf{Pow2}}(d)$
2: $\mathbf{t}_{\mathsf{Pow2}} := \mathbf{t}_{\mathsf{Pow2}} \cup (d, r)$
3: **return** r

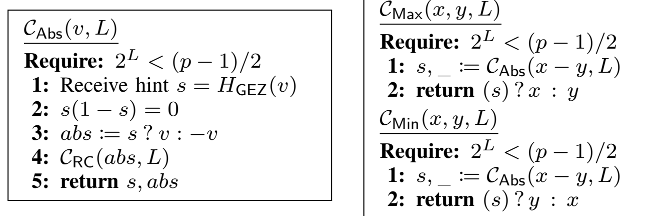
$v \ll d$

Require: $v \in [0, 2^L - 1]$, $d \in [0, K]$, $2^{L+K} < p$
1: **return** $v \cdot 2^d$

$v \gg d$

Require: $v \in [0, 2^L - 1]$, $d \in [0, K]$, $2^{L+K} < p$
1: $v' := v \cdot 2^{K-d}$
2: Receive hints $q, r = H_{\mathsf{Div}}(v', 2^K)$
3: $\mathcal{C}_{\mathsf{RC}}(q, L)$
4: $\mathcal{C}_{\mathsf{RC}}(r, K)$
5: $v' = q \cdot 2^K + r$
6: **return** q

Figure 16: Circuits for shift operations



$\mathcal{C}_{\mathsf{Abs}}(v, L)$

Require: $2^L < (p-1)/2$
1: Receive hint $s = H_{\mathsf{GEZ}}(v)$
2: $s(1 - s) = 0$
3: $abs := s\,?\,v : -v$
4: $\mathcal{C}_{\mathsf{RC}}(abs, L)$
5: **return** s, abs

$\mathcal{C}_{\mathsf{Max}}(x, y, L)$

Require: $2^L < (p-1)/2$
1: $s, _ := \mathcal{C}_{\mathsf{Abs}}(x - y, L)$
2: **return** $(s)\,?\,x : y$

$\mathcal{C}_{\mathsf{Min}}(x, y, L)$

Require: $2^L < (p-1)/2$
1: $s, _ := \mathcal{C}_{\mathsf{Abs}}(x - y, L)$
2: **return** $(s)\,?\,y : x$

Figure 15: Circuits for computing sign & absolute value ($\mathcal{C}_{\mathsf{Abs}}$) and maximum ($\mathcal{C}_{\mathsf{Max}}$) & minimum values ($\mathcal{C}_{\mathsf{Min}}$).

case where shift operations are frequently used. In order to minimize circuit size, we introduce $\mathcal{T}_{\mathsf{Pow2}}$, a lookup table for 2^d. $\mathcal{T}_{\mathsf{Pow2}}$ is first populated with entries $\mathbf{t}_{\mathsf{Pow2}} = \{(i, 2^i)\}_{i=0}^K$. Then, to compute 2^d, the prover provides a hint r, and the circuit appends (d, r) to the vector of queries $\mathbf{f}_{\mathsf{Pow2}}$, which is later enforced to satisfy $\mathbf{f}_{\mathsf{Pow2}} \in \mathbf{t}_{\mathsf{Pow2}}$. Due to the soundness of the lookup argument, r is guaranteed to be 2^d.

Assuming that $v \in [0, 2^L - 1]$ and $d \in [0, K]$ where $2^{L+K} < p$, it is straightforward to compute the left shift $v \ll d$; we only need to compute 2^d and then return $v \cdot 2^d$.

Constructing an efficient right shift gadget $v \gg d$ needs non-trivial techniques. Here, we also assume that $v \in [0, 2^L - 1]$, $d \in [0, K]$, and $2^{L+K} < p$. Naively, we could treat the right shift operation as integer division, i.e., $v \gg d = v/2^d$. The prover computes the quotient q and the remainder r such that $v = q \cdot 2^d + r$, and feeds $q, r := H_{\mathsf{Div}}(v, 2^d)$ as hints to the circuit. Then to check the predicate $P_{\mathsf{Div}}(v, 2^d, q, r)$, it is required to enforce that $q \in [0, 2^L - 1]$, i.e., $q \cdot 2^d$ does not overflow, that $r \in [0, 2^d - 1]$, i.e., the remainder should be positive and smaller than the divisor, and that $v = q \cdot 2^d + r$. We can see that checking the range of q requires decomposing an L-bit integer, and checking the range of r requires decomposing two K-bit integers r and $2^d - 1 - r$ (note that the upper bound $2^d - 1$ is a variable). This is suboptimal, as the above checks are equivalent to decomposing an $L+2K$-bit integer.

We now reduce the number of bits to be decomposed.

$\alpha \div \beta$

1: $s \coloneqq s_\alpha \oplus s_\beta$
2: $e \coloneqq e_\alpha - e_\beta$
3: Receive hints $q, r = H_{\mathsf{Div}}(m_\alpha \ll (M+2), m_\beta)$
4: $\mathcal{C}_{\mathsf{RC}}(r, M+1)$
5: $\mathcal{C}_{\mathsf{RC}}(m_\beta - r - 1, M+1)$
6: $m_\alpha \ll (M+2) = q \cdot m_\beta + r$
7: $m \coloneqq q$
8: $a \coloneqq a_\alpha \vee \mathcal{C}_{\mathsf{IsEq}}(m_\beta, 0)$
9: Receive hint $b = H_{\mathsf{MSB}}(m)$
10: $b(1-b) = 1$
11: $\mathcal{C}_{\mathsf{RC}}(m - (b \ll (M+2)), M+2)$
12: $m \coloneqq b \,?\, m : m \ll 1$
13: $e \coloneqq e + b - 1$
14: $\Delta e \coloneqq \mathcal{C}_{\mathsf{Max}}(\mathcal{C}_{\mathsf{Min}}(-2^{E-1} + 2 - e, M+2, E+1), E+1)$
15: $e', m' \coloneqq \mathcal{C}_{\mathsf{FpRound}}(e, m, \Delta e, \mathcal{C}_{\mathsf{IsEq}}(r, 0))$
16: $a' \coloneqq a \vee \mathcal{C}_{\mathsf{GEZ}}(e' - 2^{E-1}, E+1)$
17: $m'_is_0 \coloneqq \mathcal{C}_{\mathsf{IsEq}}(m', 0)$
18: $e' \coloneqq a' \,?\, 2^{E-1} : ((m'_is_0 \vee a_\beta) \,?\, -2^{E-1} + 1 - M : e')$
19: $m' \coloneqq a_\beta \,?\, 0 : (a' \,?\, 2^M : m')$
20: **return** s, e', m', a'

Figure 17: Circuit for floating-point division

Instead of naively computing $v \gg d$, the prover first computes $v' \coloneqq v \ll (K-d) = v \cdot 2^{K-d}$. Since $d \in [0, K]$, we have $K - d \in [0, K]$, and thus $v \cdot 2^{K-d} < 2^{L+K} < p$ is safe. Then we handle $v' \gg K$ analogously: the prover computes the quotient q and the remainder r for $v'/2^K$ and provides $q, r = H_{\mathsf{Div}}(v', 2^K)$ as hints, and the circuit checks the predicate $P_{\mathsf{Div}}(v', 2^K, q, r)$ by asserting that $q \in [0, 2^L - 1]$, $r \in [0, 2^K - 1]$, and $v' = q \cdot 2^K + r$. Another way to understand how to check $v' \gg K$ is that the circuit first decomposes v' into $L + K$ bits, and then computes $q = \sum_{i=0}^{L-1} 2^i v'_i$. The optimized approach only requires decomposing $L + K$ bits, saving $O(K)$ constraints.

Appendix B. Circuit for Floating-Point Division

Dividing an IEEE 754 floating-point number $\alpha = (s_\alpha, e_\alpha, m_\alpha, a_\alpha)$ by another $\beta = (s_\beta, e_\beta, m_\beta, a_\beta)$ is done in the following 4 steps — *(i)* computing the quotient of α and β, *(ii)* normalizing and *(iii)* rounding the intermediate mantissa and *(iv)* handling edge cases. We depict the corresponding in-circuit logic in Figure 17.

Compute quotient (lines 1-8). Analogous to multiplication, the quotient has sign $s \coloneqq s_\alpha \oplus s_\beta$ and exponent $e \coloneqq e_\alpha - e_\beta$. However, extra care is needed for computing the mantissa. Our rounding operation necessitates intermediate mantissas with infinite precision, but it is infeasible to represent the exact quotient when $m_\beta \nmid m_\alpha$. To address this, we instead divide α's scaled mantissa $m_\alpha \ll (M+2)$ by m_β and obtain the quotient q and remainder r, such that $m_\alpha \ll (M+2) = q \cdot m_\beta + r$, and the intermediate mantissa is $m \coloneqq q$. Since m_α, m_β are either 0 or lie in $[2^M, 2^{M+1} - 1]$, a non-zero, finite m should be bounded by $m \in (2^{M+1}, 2^{M+3})$.

The shift $M + 2$ is the smallest value that allows m to retain the correct round bit m_{M+1}, and r is used to assist the rounding process and determine the sticky bit, just as if we are rounding a mantissa with infinite precision. This is achieved by checking if r is zero. If this is the case, $m_\beta \mid (m_\alpha \ll (M+2))$, and the remaining bits after the round bit in the exact result $\frac{m_\alpha}{m_\beta}$ are all 0, implying that the sticky bit is 0. Otherwise, the sticky bit is 1.

To compute $(m_\alpha \ll (M+2))/m_\beta$ inside the circuit, the prover needs to do the division outside the circuit and provide $q, r \coloneqq H_{\mathsf{Div}}(m_\alpha \ll (M+2), m_\beta)$ as hints, and the circuit will check the predicate $P_{\mathsf{Div}}(m_\alpha \ll (M+2), m_\beta, q, r)$ by enforcing *(i)* $q \in [0, 2^{2M+3} - 1]$, *(ii)* $r \in [0, m_\beta)$, and *(iii)* $m_\alpha \ll (M+2) = q \cdot m_\beta + r$. We eliminate the check *(i)*, which is unnecessary as q's range will be narrowed to $[0, 2^{M+3} - 1]$, as we describe later. *(ii)* is converted to two range checks since the upper bound m_β is a variable.

The quotient is abnormal if the dividend is abnormal or the divisor is ± 0 or NaN, i.e., $a \coloneqq a_\alpha \vee \mathcal{C}_{\mathsf{IsEq}}(m_\beta, 0)$.

Normalize intermediate mantissa (lines 9-13). m is normalized in the same way as the normalization of multiplication. Since a non-zero and finite m is in $(2^{M+1}, 2^{M+3})$, the leading 1 of a non-zero m is either the $M+1$-th bit or the $M+2$-th bit, and we check if $m_{M+2} = 1$. If so, m and e are unchanged. Otherwise, $m \coloneqq m \ll 1, e \coloneqq e - 1$, where e is decremented as $m_{M+2} = 0$ indicates that the division borrows. The in-circuit operation is similar to normalization for multiplication. The prover feeds $b \coloneqq H_{\mathsf{MSB}}(m) = m_{M+2}$, the MSB of m, as a hint to circuit, and the circuit checks the predicate $P_{\mathsf{MSB}}(m, b)$ in 2 steps: *(i)* enforce b is a boolean, and *(ii)* assert $m - (b \ll (M+2)) \in [0, 2^{M+2})$. Note that *(ii)* implies that $m \in [0, 2^{M+3} - 1]$, which indicates the hinted $m \coloneqq q$ is the correct quotient mantissa and thus lies in $(2^{M+1}, 2^{M+3})$. Finally, the circuit updates m, e according to b, i.e., $m \coloneqq b \,?\, m : m \ll 1$, $e \coloneqq e + b$.

Round intermediate mantissa (lines 14-15). The normalized mantissa m of length $N = M+3$ is rounded as in Section 3.2, with $\Delta e = \max(\min(-2^{E-1} + 2 - e, K), 0), K = M+2$, obtaining e' and m'. In addition, the equality between r and 0 is used to determine the sticky bit, thus we set the in-circuit parameter $aux \coloneqq \mathcal{C}_{\mathsf{IsEq}}(r, 0)$.

Edge Cases (lines 16-20). We omit the detailed explanation due to space limit.

Appendix C. Authentic Location Information

Currently, the ZKLP paradigm described in Section 4 only introduces efficient circuits for transforming (θ, ϕ) to (i, j, k) in the Uber H3 [17] hexagonal spatial index. Whilst (θ, ϕ) are private inputs, and hence not disclosed to the verifier, the prover could still choose arbitrary values as input to the circuit, as it doesn't constrain that location information is obtained correctly, i.e., it does not ensure *data provenance*. We introduce three approaches to mitigate this issue by proving that data comes from a trusted source.

(i) Offline Finding Networks Offline Finding ecosystems, such as Apple's "Find My" network, allow device owners to track the location of missing offline devices via Bluetooth, and report an approximated location via the internet [12], [54]. Each device within these networks generates

$\mathcal{C}_{\mathsf{IJK}}(x, y)$

1: $a_1 := |x|\,;\;\; a_2 := |y|$
2: $x_2 := \frac{a_2}{\sin(\frac{\pi}{3})}\,;\;\; x_1 := a_1 + \frac{x_2}{2}$
3: $m_1 := \lfloor x_1 \rfloor\,;\;\; m_2 := \lfloor x_2 \rfloor$
4: $r_1 := x_1 - m_1\,;\;\; r_2 := x_2 - m_2$
5: $r_{1,\mathrm{A}} = (r_1 < \frac{1}{2})\,?\,1 : 0\,;\;\; r_{1,\mathrm{A1}} = (r_1 < \frac{1}{3})\,?\,1 : 0$
6: $r_{1,\mathrm{B1}} = (r_1 < \frac{2}{3})\,?\,1 : 0$
7: $i_{\mathrm{A2},1} = (1 - r_1 \leq r_2)\,?\,1 : 0\,;\;\; i_{\mathrm{A2},2} = (2 \cdot r_1 > r_2)\,?\,1 : 0$
8: $i_{\mathrm{B1},1} = (r_2 > 2 \cdot r_1 - 1)\,?\,1 : 0\,;\;\; i_{\mathrm{B1},2} = (1 - r_1 > r_2)\,?\,1 : 0$
9: $i_{\mathrm{A}} = (i_{\mathrm{A2},2}\,?\,m_1 + 1 : m_1)$
10: $i_{\mathrm{B}} = (i_{\mathrm{B1},1}\,?\,(i_{\mathrm{B1},2}\,?\,m_1 : m_1 + 1) : m_1 + 1)$
11: $i = r_{1,\mathrm{A}}\,?\,(r_{1,\mathrm{A1}}\,?\,m_1 : i_{\mathrm{A}}) : (r_{1,\mathrm{B1}}\,?\,i_{\mathrm{B}} : m_1 + 1)$
12: $j_A = (\frac{r_1+1}{2} > r_2)\,?\,1 : 0\,;\;\; j_B = (1 - r_1 > r_2)\,?\,1 : 0$
13: $j_C = (\frac{r_1}{2} > r_2)\,?\,1 : 0$
14: $j_{\mathrm{A}} = r_{1,\mathrm{A1}}\,?\,((j_A)\,?\,m_2 : m_2 + 1) : ((j_B)\,?\,m_2 : m_2 + 1)$
15: $j_{\mathrm{B}} = r_{1,\mathrm{B1}}\,?\,((j_B)\,?\,m_2 : m_2 + 1) : ((j_C)\,?\,m_2 : m_2 + 1)$
16: $j = r_{1,\mathrm{A}}\,?\,j_{\mathrm{A}} : j_{\mathrm{B}}$
17: $i_{>} = (j < i)\,?\,1 : 0$
18: $i_{-} = (x < 0)\,?\,((y < 0)\,?\,1 : i_{>}) : ((y < 0)\,?\,(1 - i_{>}) : 0)$
19: $i_A = (y < 0)?(i_{>}?(i{-}j) : (j{-}i)) : -i;\; i_B = (i_{>}?(i{-}j) : (j{-}i))$
20: $i = (x < 0)\,?\,i_A : ((y < 0)\,?\,i_B : i)$
21: **return** $\mathcal{C}_{\mathsf{Normalize}}(i_{-}, i, y.S, j, 0, 0)$

Figure 18: Sub-Circuit for computing the conversion of two dimensional hexagon coordinates x, y to three dimensional coordinates i, j, k. Primitive Operations are floating-point.

$\mathcal{C}_{\mathsf{Normalize}}(i_{-}, i, j_{-}, j, k_{-}, k)$

1: $i_{>j} = (j < i)\,?\,1 : 0\,;\;\; i_{>k} = (k < i)\,?\,1 : 0$
2: $j_A = (j_{-})\,?\,((i_{>j})\,?\,(i - j) : (j - i)) : (i + j);$
3: $j_{A-} = (j_{-})\,?\,(1 - i_{>j}) : 0$
4: $k_A = (k_{-})\,?\,((i_{>k})\,?\,(i - k) : (k - i)) : (i + k);$
5: $k_{A-} = (k_{-})\,?\,(1 - i_{>k}) : 0$
6: $i = (i_{-})\,?\,0 : i$
7: $j = (i_{-})\,?\,j_A : j\,;\;\; j_{-} = (i_{-})\,?\,j_{A-} : j_{-}$
8: $k = (i_{-})\,?\,k_A : k\,;\;\; k_{-} = (i_{-})\,?\,k_{A-} : k_{-}$
9: $i = (j_{-})\,?\,(i + j) : i\,;\;\; j = (j_{-})\,?\,0 : j\,;\;\; k = (j_{-})\,?\,k_A : k$
10: $k_{-} = (j_{-})\,?\,k_{A-} : k_{-}$
11: $i = (k_{-})\,?\,(i + k) : i\,;\;\; j = (k_{-})\,?\,(j + k) : j\,;\;\; k = (k_{-})\,?\,0 : k$
12: $\min = (i_{>j})\,?\,j : i$
13: $\min = (k < \min)\,?\,k : \min$
14: $i = i - \min\,;\;\; j = j - \min\,;\;\; k = k - \min$
15: **return** $[i, j, k]$

Figure 19: Normalization Sub-Circuit for adjusting i, j, k coordinates. Primitive Operations are floating-point.

unique public-private key pairs and frequently rotates its public keys to mitigate tracking risks. Lost devices broadcast their public key, which nearby Apple devices utilize to encrypt their own location. The encrypted location data is then sent to and stored on Apple's servers and can only be decrypted by device owner's private key. Although these ecosystems is proprietary to the manufacturer, recent work shows how to utilize these offline finding protocols (e.g., Apple's "Find My" network) to localize arbitrary devices [12].

To obtain authentic location information, one can leverage the network of unknown devices that post "Location Reports" to Apple's server. A "Location Report" can only be decrypted by the owner of the private key using AES-128-GCM. After decryption, the location data (latitude and longitude) is available in plain. To ensure authenticity and prevent forgery, one can prove the correct key derivation and decryption of several location reports, and further prove correct triangulation before applying the optimized ZKLP circuits. Given recent work that shows how to optimize non-native field arithmetic in SNARKs with lookup arguments [55], this solution effectively bridges the cyber-physical gap without demanding for additional hardware or distributed networks that may not yet be in place.

(ii) Authenticated GNSS signals. Traditionally, GNSS services were for military use, which meant that they lacked robust security features like signal authentication. Addressing this legacy issue necessitates modifications to millions of GPS receivers. Recently, the security vulnerabilities in GPS have gained attention, leading to efforts to improve its security. One such effort is Open Service Navigation Message Authentication (OSNMA), which targets the lack of signal authentication [11]. Open-source implementations of OSNMA receivers exist and are well-documented [56].

OSNMA combines ECDSA signatures, the Timed Efficient Stream Loss-tolerant Authentication (TESLA) key chain mechanism, and Messages Authentication Codes (MACs) for message authentication. The receiver's cryptographic operations in OSNMA include verifying a root key of the TESLA chain, authenticating new public keys, verifying TESLA chain keys, and authenticating the MACs of navigation messages. First, the receiver validates the authenticity of a Root Key through an ECDSA signature. Successively, the receiver uses the authenticated root key to verify the current chain key. By successfully verifying the chain key against the root key, the receiver ensures that the chain key is part of the legitimate TESLA key chain. Using the verified chain key, the receiver computes the MAC of the navigation data. The computed MAC is compared with the extracted MAC. If they match, it confirms that the navigation message is authentic and has not been tampered with.

We presume that the overhead of proving an authenticated GNSS signals with OSNMA will be dominated by the in-circuit verification of ECDSA signatures. At the time of writing, it requires $4 \cdot 10^6$ constraints for in-circuit emulation over the circuit unfriendly curve secp256k1 in gnark. Further, there remains an open problem — a malicious prover may collaborate with a third party, which obtains the GNSS signal and forwards it. The verifier needs to ensure that *(i)* obtaining the location and *(ii)* computing the proof is conducted *atomically* — which remains unsolved.

(iii) TLS Oracles. Alternatively, one may trust a third party entity to verify that location information is obtained from a trusted entity. TLS Oracles [57], [58], [59], [60] provide the possibility to verify data provenance by extending a plain TLS session with a third party, which verifies that the data obtained by a client from a server is authentic. As such, one could extend the ZKLP paradigm to obtain location information from an API endpoint. The request includes information about nearby cell towers and WiFi access points detected by a mobile client, which allows for accurate location estimation via an external API. We leave the details of potential optimizations for authenticated and atomic GNSS, and a concrete system design for integration with TLS oracles, to potential future work.