Succinct Zero Knowledge for Floating Point Computations

Sanjam Garg
sanjamg@berkeley.edu
UC Berkeley and NTT Research
Berkeley, California, USA

Abhishek Jain
abhishek@cs.jhu.edu
Johns Hopkins University
Baltimore, Maryland, USA

Zhengzhong Jin
zjin12@jhu.edu
Johns Hopkins University
Baltimore, Maryland, USA

Yinuo Zhang
yinuo.yz@gmail.com
UC Berkeley
Berkeley, California, USA

ABSTRACT

We study the problem of constructing succinct zero knowledge proof systems for *floating point computations*. The standard approach to handle floating point computations requires conversion to binary circuits, following the IEEE-754 floating point standard. This approach incurs a $\mathrm{poly}(w)$ overhead in prover efficiency for computations with w-bit precision, resulting in very high prover runtimes – already the key bottleneck in the design of succinct arguments.

We make the following contributions:

- We propose a new model for verifying floating point computations that guarantees approximate correctness w.r.t. a relative error bound. This model is inspired by numerical analysis, and is very meaningful for applications such as machine learning and scientific computing.
- Using this model, we present a general method for constructing succinct zero-knowledge proofs for floating point computations starting from existing public-coin "commit-and-prove" systems. For computations with w-bit precision, our approach incurs only a $\log(w)$ overhead in prover running time. Our compiler nearly preserves (up to a factor of 2) the communication complexity of the underlying protocol, and requires sub-linear verification time. The resulting proof can be made non-interactive in the random oracle model. Concretely, our scheme is $\sim 57\times$ faster than the method following IEEE standard exactly [35] for 32-bit floating point computations.

Central to our main result, and of independent interest, is a new batch range proof system in standard prime order groups that does *not* rely on bit decomposition.

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11.
https://doi.org/10.1145/3548606.3560653

CCS CONCEPTS

• **Security and privacy → Mathematical foundations of cryptography**.

KEYWORDS

Succinct Proof System; Zero-knowledge; Verifiable Computation.

ACM Reference Format:
Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Yinuo Zhang. 2022. Succinct Zero Knowledge for Floating Point Computations. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3560653

1 INTRODUCTION

Succinct proofs [24, 26] allow a prover to convince a verifier that an $\mathcal{NP}$ statement is true, with communication sub-linear in the size of the prover's witness. Such proofs are studied in two avatars: interactive proofs, where prover and the verifier communicate over multiple rounds, and non-interactive proofs, where the prover sends a single message to the verifier. If the interactive proof is public-coin, known methods (e.g., [17]) can be used to transform it into a non-interactive proof in the random oracle model [3], or the common reference string model.

Succinct proofs are typically paired with an additional *zero knowledge* (ZK) property [21], which requires that the verifier does not learn anything beyond the veracity of the statement. In recent years, succinct zero-knowledge proofs have found numerous real-world applications, e.g., in the design of blockchains [4]. This has led to extensive research towards improving the efficiency of succinct zero-knowledge proofs in practice [1, 5, 9, 10, 13, 14, 20, 29, 32, 36–38], both in terms of prover and verifier running times. The standard approach in all of these works is to model the computation as an arithmetic or binary circuit, where each wire of the circuit is represented as an *integer* value.

How to Verify Floating Point Computations? In many real-world applications, *floating point* computations are ubiquitous. A floating point number is different from an integer, in that it is expressed using an integer *significand* part (with a fixed number of digits) and scaled using an integer *exponent* part (in a fixed base). The parameters of machine learning models are usually stored as floating point numbers in computers. In physics, any measurement we make may incur some inaccuracy, and hence the result is usually represented as a floating point number.

In this work, we study the following question,

Can we build efficient succinct zero-knowledge proofs for floating point computations?

Current Approach and Drawbacks. The standard approach to prove the correctness of a floating point computation is to first convert the floating point operations to binary circuits, following the IEEE-754 floating point standard (see, e.g. [35]). The IEEE standard guarantees that the result of each floating point operation is rounded to the closest floating point number, and except for some corner cases, the result of computation is almost deterministic.

However, this method introduces a very high computational overhead. Asymptotically, for floating point numbers with w-bit precision, *each* operation requires a binary circuit of size *polynomial* in w. As a concrete example, [35] converts multiplication operation over IEEE 32-bit floating point numbers with precision $w = 24$ into a binary circuit of 8854 gates. This means that the prover running time – the key bottleneck in the most efficient known constructions of succinct proofs systems – increases roughly 9000-fold.

Our Work. In this work, we aim to build succinct zero-knowledge proof systems with improved prover efficiency. Inspired by *numerical analysis* [33], we propose a new model for proving the correctness of floating point computations. In our model, we only verify an upper bound on the *relative error* for each floating point operation in the overall computation. As we discuss shortly, our model provides strong guarantees for most applications in machine learning and scientific computing. Crucially, it allows us to avoid the overhead of transforming floating point operations to binary circuits, and build proof systems with significantly better prover efficiency.

Our work is inspired by (although different from) an influential recent line of research [6, 12] that constructs efficient fully homomorphic encryption [18] schemes for floating point computations by deviating from the IEEE standard.

2 OUR CONTRIBUTIONS

In this work, we initiate the study of *prover- efficient* succinct zero-knowledge proofs for floating point computations. We start by providing a summary of our contributions.

- We first propose a relative error model to verify floating point computations.
- We build a generic compiler that transforms any (public-coin) "commit-and-prove" zero-knowledge proof of knowledge system into a succinct zero-knowledge proof system for floating-point computation. For computations with w-bit precision, the prover time in our system grows only *logarithmically* in w. The communication complexity nearly preserves the communication of the underlying protocol (up to a factor of 2), and the verification time is sub-linear in the size of the computation.
- Finally, we provide a concrete efficiency analysis of our protocol. Compared to the prior method that involves verifying that the computation followed the IEEE standard exactly [35], the prover running time of our scheme is $\sim 57\times$ faster for 32-bit floating point numbers, and $236\times$ faster for 64-bit floating point numbers. We also compare performance with an alternative solution that uses optimized versions of existing range proofs (that rely on bit-decomposition) within our relative error model. In this case, the prover runtime of our scheme is $\sim 2.5\times$ to $3.7\times$ faster, depending on the precision. Our improvements are much more significant if multiplication gates dominate the relation circuit; see Section 2.2 and 6.7 for more details.

We now describe our contributions in more detail. We first provide an overview of our model in Section 2.1. We then describe our results and performance analysis in Section 2.2.

2.1 Our Model

We introduce the following model for proving the approximate correctness of floating point computations: a (honest) prover performs the floating point computation following the IEEE standard, and then proves an upper bound on the *relative error* for each step of the floating point computation. Specifically, for each (addition or multiplication) gate g with input wires a, b and output wire c, we require a prover to prove that

$$|c - g(a, b)| \leq \delta |g(a, b)|,$$

for a relative error bound δ, where $g(a, b)$ is the *precise* value of the addition (resp. multiplication). Here δ can be set as machine epsilon, which is the relative error bound in the IEEE standard. Zeros, infinities, and not-a-number in IEEE standard can be treated separately as corner cases.

Our model is directly inspired by the field of *numerical analysis* [33] that concerns with measuring the accuracy of numerical algorithms. A general methodology is to first bound the relative error for each step of the floating point computation, and then an error bound on the output is derived. The relative error methodology is used because the behavior of the rounding operation is quite complicated and hard to reason about. The accuracy of most numerical algorithms can be analyzed in this way.

Meaningfulness of our Model. For any floating point computation program, a proof in our model can be combined with numerical analysis about the program to obtain guarantees on the *accuracy of the output* of the program. For example, in numerical linear algebra [34], the backward error analysis for basic numerical algorithms such as Gaussian elimination and matrix decomposition relies *only* on the bound of relative errors. These algorithms serve as a basis for numerous scientific computing tasks ranging from numerical differential equation solving to big data analysis [7]. For these applications, strong guarantees on the accuracy of the output can be obtained using our approach.

What about more complicated floating point programs for which rigorous numerical analysis results are not known? The output of such programs is at least robust in the presence of rounding errors that arise in the computation following the IEEE standard. Hence, we can heuristically assume that their output is robust to any small perturbations bounded by the relative error upper bound during the computation. Indeed, a counterexample for our assumption would imply that the accuracy of the output cannot be argued by following the relative error methodology in numerical analysis but the program is "numerically stable" in practice. This would greatly

advance our understanding of the numerical algorithm being used and the relative error methodology in numerical analysis.

Finally, we note that the adversarial nature of proof systems diminishes any error gap between our model and the alternative model of proving that a computation was performed following the IEEE standard [35]. As an example, our model does not guarantee that the value in the output wire of a step of the computation is rounded to the nearest number (as is done in the IEEE standard). Instead, our model allows a larger (by a factor of 2 in some cases)[1] *absolute* error. Note, however, that such difference only appears when the floating point numbers are public and fixed. In the setting of zero-knowledge proofs, the floating point numbers are *hidden* to the verifier, and a malicious prover may choose any floating point numbers that lead to the maximum possible rounding error in the worst case. Hence, the actual rounding error matches the upper bound in the relative error model.

Efficiency Benefits. Our model of verifying relative errors enables us to achieve *better prover efficiency* than existing methods. In particular, our model does not require proving complicated rounding operations. Instead, it only requires proving "simple" relations between a tuple of wire values (a, b, c) associated with an operation. Furthermore, as we discuss below, we can build direct proofs for such relations without using bit decomposition.

2.2 Our Constructions

We construct *succinct* zero-knowledge proofs for floating point computations, both in the interactive and non-interactive setting. Our main result is a generic compiler that compiles any *commit-and-prove* succinct ZK proof of knowledge system into a succinct ZK proof for floating point computations in our model.

Recall that a commit-and-prove (CP) system is an interactive protocol that allows a prover to prove that a committed value is a witness of some instance in an $\mathcal{NP}$ language. Most existing succinct ZK proof systems (see, e.g. [5, 9, 29, 36]) can be abstracted as CP systems. We require such proof systems for R1CS – an already popular choice of $\mathcal{NP}$ complete language in existing systems.

General Compiler. Let w be the precision of floating point numbers, k be the bit-length of the exponent, and C be a floating point computation circuit. Our general compiler yields a protocol with the following parameters:

- **Prover Time:** The run-time of the prover is equivalent to the prover run-time in the underlying CP protocol for proving an R1CS instance $\mathbb{X}$ of size $|\mathbb{X}| = O(\log w + k) \cdot |C|$.
- **Proof Size:** The proof size is equivalent to the proof size of the underlying CP protocol for proving an R1CS instance of size $|\mathbb{X}|$, with $O(1)$ additional field elements.
- **Verification Time:** The verification time is equivalent to the verification time of the underlying CP protocol for $|\mathbb{X}|$-size R1CS, with an additional $O(\sqrt{|\mathbb{X}|})$ group operations.

If the underlying CP system is *public-coin* and *succinct*, then so is our resulting protocol. Furthermore, by relying on the *zero knowledge* property of the underlying CP systems, we also achieve zero knowledge property. Thus, instantiating our compiler with a (public-coin) commit-and-prove succinct ZK proof (of knowledge) system, we obtain a (public-coin) succinct ZK proof system for floating point computations. Applying the Fiat-Shamir transformation [3] to our protocol, we obtain a *non-interactive* succinct ZK argument system for floating point computations.

The verification time of our proof system is *sub-linear* in the circuit size, if the underlying CP protocol achieves sub-linear verification. More precisely, our protocol incurs an additive overhead of $O(\sqrt{|C|})$ in the verification time of the underlying CP protocol. While it is clearly desirable to achieve smaller verification time, we note that trading verification time for better prover performance has been an active area of recent research (see, e.g., [1, 5, 9]). Such trade-offs can be justified by practical ramifications; indeed, some of these proof systems are being used in real-world systems.

Our technique for achieving sub-linear verification can be naturally extended to achieve poly-logarithmic verification. This, however, results in an increase in prover running time (due to the necessity of a larger group; see Section 6.3). We leave open the problem of achieving poly-logarithmic verification time (without blowing up the prover running time) for future research.

Concrete Parameters. For a concrete comparison of our approach with the IEEE-754 standard and the optimized bit-decomposition method (see Section 6.7 for its detailed description), we list the parameters of zero-knowledge CP protocol regarding the three different approaches, for 32-bit floating point numbers ($w = 24, k = 8$) and 64-bit floating point numbers ($w = 53, k = 11$) in Table 1. The parameters support floating-point circuits of $\approx 2^{20}$ gates. While our construction supports different moduli, here we choose the modulus $p \approx 2^{384}$, which can be used with 384-bit elliptic curves BN-384.

The R1CS size of our protocol is $91\times \sim 432\times$ smaller than following the IEEE standard exactly and $4\times \sim 7\times$ smaller than the optimized bit decomposition method, depending on the length of floating point number (32 bits or 64 bits). If we use the 384-bit groups for all methods, then such improvement translated to the running time of the prover efficiency. Since the optimized bit-decomposition method can also use smaller groups such as 256-bit groups, taking into account the difference in group sizes, our protocol is still $57\times \sim 236\times$ faster than following IEEE standard and $2.5\times \sim 3.7\times$ faster than the optimized bit-composition method in terms of prover efficiency. We achieve these improvement at the cost of doubling the proof size. Finally, we note that our construction achieves significantly more improvement for the case of multiplication gates (as opposed to addition gates); hence, if the circuit is dominated by multiplication gates, our overall improvements will be more significant. For more details in performance estimation and comparison, see Section 6.7.

Batch Range Proofs. Central to our main result is a new construction of *batch range proof* system in groups of (known) prime order, without relying on bit-decomposition techniques.

A batch range proof system allows for proving an instance of the form $\{(y_i, [\ell_i, r_i])\}_i$, where y_i, ℓ_i, r_i are committed using some commitment scheme. The prover tries to convince a verifier that for every i, y_i is in the interval $[\ell_i, r_i]$.

Implicit in our main result is a public-coin succinct batch range proofs from any public-coin succinct "commit-and-prove" proof

[1] The upper bound of the relative error caused by *rounding to nearest* rule is in the range $[\delta/2, \delta]$, where δ is machine epsilon. See [19].

		This work	Bit.	IEEE-754
32-bit	|R1CS| per (+)	89	296	2456
	|R1CS| per (×)	35	207	8854
	Overall proof size	$2(\|\Pi\| + \|c\|)$	$\|\Pi\| + \|\mathbf{c}\|$	$\|\Pi\| + \|\mathbf{c}\|$
64-bit	|R1CS| per (+)	115	528	15637
	|R1CS| per (×)	24	439	44899
	Overall proof size	$2(\|\Pi\| + \|c\|)$	$\|\Pi\| + \|\mathbf{c}\|$	$\|\Pi\| + \|\mathbf{c}\|$

Table 1: Concrete performance for floating point number addition/multiplication, where we choose $\log_2 p = 384$. "IEEE-754" refers to the method of converting a floating point addition/multiplication to a binary circuit. For 32-bit floating numbers we list the circuit size in the source code of [35]. For 64-bit floating numbers we list the circuit size achieved by [2]. The "Bit." refers to the bit decomposition method used by [9] for range proofs, optimized for floating point computations. |R1CS| refers to the size of R1CS instance for the underlying protocol. The numbers under "This work" correspond to our succinct zero-knowledge protocol with linear verification time. "Overall Proof size" is the size of the prover's message size, where $|\Pi|$ is the proof size of the underlying "commit-and-prove" proof system and $|c|$ is the size of the underlying commitment.

system for R1CS over a prime order field $\mathbb{F}_p$. Our batch range proof with the following properties.

- **Prover Time:** The prover's running time for a R1CS instance $\mathbb{X}$ with $|\mathbb{X}| = O(\log w + k) \cdot n$, where n is the number of instances in the batch range proof.
- **Proof Size:** The underlying proof with $O(1)$ additional group or field elements.
- **Verification Time:** The verification time of the underlying "commit-and-prove" for $\mathbb{X}$, with additional $O(|\mathbb{X}|^{1/2})$ group operations.

Known constructions of batch range proofs fall into two categories: the first approach relies on bit decomposition [9, 11, 23], which introduces an $\Omega(w)$ overhead in prover time for proving that a w-bit integer is within some range. For example, for 32-bit floating point number, the bit-decomposition method needs 3.3× larger R1CS for each addition gate, and 5.9× larger R1CS for each multiplication gate.

Another approach relies on groups of unknown order [8, 22], which is computationally inefficient[2]. Our construction, in contrast, relies on prime (known) order groups and does not require bit decomposition. This result might be of independent interest.

2.3 Related Work

To the best of our knowledge, there is no prior work on succinct proof systems that supports full functionality of floating point computations.

Weng et al. [35] proposed a (non-succinct) ZK proof system that supports floating point computation. Their approach involves compiling the floating point computation to a binary circuit following the IEEE standard. Their implementation only supports single precision (32-bit) floating point computation.

Setty et al. [31] proposed a general proof system that supports integer and rational number arithmetic. They also support floating point computation by rational numbers. However, they do not support *rounding operation*, which is crucial for floating point computations in practice.

3 TECHNICAL OVERVIEW

In this section, we provide an overview of our techniques. To simplify our illustration, we first only consider fixed point computation, where all the wire values can be essentially viewed as w-bits integers (for fixed point number with precision w). We will extend our ideas to floating point computations later on in this overview.

Firstly, we show how to prove that the relative error for an addition gate is small. In other words, we want to prove the following inequality:

$$|a + b - c| < \delta|a + b|,$$

where a, b are input wires to the addition gate and c is the output wire, all being w-bits integers (up to an 2^{-w} factor that can be dropped all together). Note that for each gate, we can write an inequality as above. Hence there will be a *batch* of inequalities that we want to prove. To build succinct proofs for them, our starting point is the recent work [15], where they build a range proof in known order groups without bit decomposition, but without support for batching.

Following a line of research [16, 22, 25] on range proofs, their idea is to first turn each inequality into the compatible form of a range proof: $z > 0$. This can be done by adding some intermediate constraints and variables. When it comes to prove that $z > 0$, we turn to Legendre's three square theorem, which states that any positive integer that equals to 1 mod 4 can be expressed as the sum of three squares. Specifically, $(z > 0) \Leftrightarrow (4z - 3 > 0)$, and there always exists three integers $\{\gamma_i\}_{i\in[3]}$ such that

$$4z - 3 = \sum_{i=1}^{3} \gamma_i^2. \tag{1}$$

Furthermore, these three integers can be found efficiently in $O(\log^2 z)$ time [28]. Hence, to prove $z > 0$, one only needs to provide $\{\gamma_i\}$'s and show that $4z - 3$ is the three square sum of them.

Nevertheless, it does not work directly: Recall that for most of the existing succinct proofs, usually the prover firstly uses a succinct commitment to commit the wires, and then interacts with the verifier to ensure that the committed values satisfy the required constraint using some PCP-based method. In this framework, the prover can only prove statements in the finite field, either because the commitment scheme is group based, or the PCPs need to work in a finite field. Consequently, the soundness of such protocols can only ensure that Equation 1 holds in some finite field. Due to the wrap around in the finite field, it could be that the Equation 1 holds modulo some number p, but not holds over integers.

To circumvent this, one possible direction is to use an unknown order group, one candidate being the RSA group [22]. However, the RSA group size is relatively larger than the known order groups

[2]The typical choices of unknown order groups are RSA groups and class groups. Both of them need large group size to resist subexponential time attacks.

such as elliptic curves. Another direction could be using class groups. But those groups are computationally inefficient as well [30]. Hence, we set our goal as building such succinct proofs in standard prime known-order groups.

This Work. To overcome the aforementioned barrier, our idea is to further ensure that all the values z and γ_i's are very small compared to p, so that the wrapping around in modulo p fields does not happen. One naive way to achieve this is to have the verifier query each value, but this requires the prover to open commitment of each wires, thus leading to *non-succinct* proofs.

Achieving Succinctness. To resolve this issue, we extend the idea of *random linear combinations*. Simply describing, to prove that a batch of values y_i's are all 0, one could have the verifier to send some random coefficients $r_i \leftarrow \mathbb{F}_p$, and have the prover prove that $\sum_i y_i \cdot r_i = 0$.

In our case, we want to prove all y_i's are small, but we can't use random linear combination directly. Because if one of y_i is not zero, then the random linear combination $\sum_i y_i \cdot r_i$ is uniformly random in $\mathbb{F}_p$, which tells nothing about whether all y_i's are small. Hence, instead of sampling r_i's randomly in $\mathbb{F}_p$, we sample them in a small range. In this way, the summation $\sum_i y_i \cdot r_i$ should also be small. Then we have the verifier check whether the summation is also in a small range or not. To argue soundness, we hope to prove that if one of y_i is large, then the random linear combination is also large with overwhelming probability.

However, the above statement doesn't hold. In fact, there is a simple counterexample. Consider the case where there is only one element $y_1 = 2^{-1} \pmod p$. Then $y_1 = (p+1)/2$ is a large value for any prime $p > 2$. However, if we sample r_1 from a small range, then with probability $1/2$, r_1 is an even number. Then $y_1 \cdot r_1 = (r_1/2) \pmod p$, which is a small value. This counterexample can be extended to more general case where each y_i is a "fraction". For more detail, see Section 6.6.

Argue Soundness. Using a careful analysis, we can prove that such counterexamples are the *only possible* counterexamples. Namely, let's consider y_i's such that $\Pr_{\{r_i\}_i}[\sum_i y_i \cdot r_i \bmod p \text{ is small}] > 1/\mathsf{poly}(\lambda)$, where each r_i is sampled from a small range. Then we can prove that each y_i must be of the form $A_i/B_i \pmod p$, where both A_i's and B_i's are small integers. To prove this, notice that for each index i, by an averaging argument, there must exist a series of r_j's where $j \neq i$, such that conditioning on them, we have $\Pr_{r_i}[y_i \cdot r_i + \sum_{j\neq i} y_j \cdot r_j \text{ is small} \mid \{r_j\}_{j\neq i}] > 1/\mathsf{poly}(\lambda)$, where the randomness is only over the i-th coordinate r_i. Now, if we set the range of r_i to be super-polynomial, then by counting argument there must exist two different r_i, r_i' such that their random linear combination with y_i's are both small. If we denote the random linear combinations of y_i with respect to r_i and r_i' as Y_i and Y_i', then we have $y_i = (Y_i - Y_i') \cdot (r_i - r_i')^{-1} \pmod p$.

In this way, we prove that each y_i is in a form of "fractions", where both the numerator and denominator are small integers. Towards arguing soundness, we need to resolve the following two challenges:

- The Equation (1) holds only modulo p. We need to drop the modulo operation so that we can derive $z > 0$ from Equation (1).
- Y_i's are in the form of "fractions". We need them to be integers.

To address the first challenge, we observe that, if we set the modulo p to be large enough, then each equation for "fractions" over $\mathbb{F}_p$ implies that there exists an assignment of the variables over *real numbers* such that the Equation (1) holds without modulo p. To see this, consider an addition equation $a_1 + a_2 = a_3 \pmod p$, where $a_1 = A_1 \cdot B_1^{-1}, a_2 = A_2 \cdot B_2^{-1}, a_3 = A_3 \cdot B_3^{-1} \pmod p$ are all "fractions" with small numerators and denominators. Multiplying both sides by $B_1B_2B_3$, we have $A_1B_2B_3 + A_2B_1B_3 = A_3B_1B_2 \pmod p$.

Since A_i's and B_i's are small integers, then for a relatively large p, the modulo p operation does not wrap around. Hence, the same equation holds over integers and we have $A_1/B_1 + A_2/B_2 = A_3/B_3$ over reals, without modulo p.

Back to the previous example of checking an addition gate: assume that using random linear combination test, we identify some fractional numbers z^* and γ_i^* such that the following inequality holds over $\pmod p$: $4z^* - 3 = \sum_{i=1}^{3} \gamma_i^{*2}$. Deploying above argument, we can see that this inequality also holds over the reals, thus the fraction $z^* > 0$, as desired. This further implies that there exists input wire values $a^*, b^* \in \mathbb{R}$ and output wire value $c^* \in \mathbb{R}$, such that this addition gate has small relative error with respect to these values (details in the full version This argument can also be extended to all multiplication gates, thus we argue that there exists an assignment of real numbers to all the wires such that all gates in the circuit have small relative errors.

For the second challenge, instead of further arguing Equation 1 holds over integers rather than reals, we define soundness for the following weaker notion. If there doesn't exist an assignment of the wire values in real numbers such that each gate is correct up to some relative error δ, then any cheating proof will be rejected. We note that there is a theoretical gap between the completeness and soundness properties. Recall that, completeness requires that if the circuit can be approximately satisfied by a set of floating point numbers of some fixed precision w, then the honest prover should be accepted. However, it's possible that a circuit is not approximately satisfied by floating point numbers, but is approximately satisfiable in real numbers. However, we expect this gap to be rather narrow in practice. Intuitively, for any floating point circuit that is robust to small perturbation caused by rounding errors, if the circuit is approximately satisfiable over reals then we can take the precision to be slightly larger, such that those real numbers can be rounded to floating point numbers, and those floating point numbers can make the circuit approximately satisfiable.

Summary (So Far). We now give a summary of our construction (so far).

- Firstly, we convert a fixed point circuit to an R1CS instance, containing all necessary constraints such as Equation 1.
- **Prover:** Commit the R1CS witness and send the commitment to the verifier.
- **Verifier:** Send the random linear combination coefficients $\{r_i\}_i$, where r_i's are small.
- **Prover:** Compute the random linear combination $\{v_j\}_j$ and send it to the verifier. The prover also uses the underlying commit-and-prove proof system to prove that the witness

satisfies the R1CS instance, and the random linear combination is computed correctly.
- **Verifier:** The verifier verifies the proof, and checks whether the random linear combinations $\{v_j\}_j$ are in a small range.

If we use the above construction directly, we need to choose a modulo p that is large enough for all wires values. However, we note that the wire values can have different orders of magnitude. Hence, in our actual construction, we split the variables to two disjoint sets according to their magnitudes, and use two random linear combinations to test them separately. In this way, we can choose a smaller modulo p. There are several more optimizations in our construction. For more details, see Section 6.1.

Next, we explain how we extend our techniques to handle floating point computations.

Extension to Floating Point Numbers. A *floating point number* of precision w differs from a fixed point number of precision w in that it has an additional part containing exponents. More formally, any floating point number can be written as $s \cdot 2^{e-w}$, where s is a w-bit integer with most significant bit fixed to 1 (hence normalized), and e is a k-bit integer. When all the wires are given as floating point numbers, checking the relative error for each gate becomes trickier, since their exponent parts play a role in the inequality as well.

Let us start with floating number multiplication: we want to check the following essential inequality; $|s_a \cdot 2^{e_a} \cdot s_b \cdot 2^{e_b} - s_c \cdot 2^{e_c}| \leq \delta \cdot |s_a \cdot 2^{e_a} \cdot s_b \cdot 2^{e_b}|$.

Clearly, one can reduce this task to that of fixed point multiplication, by asking the prover to supply intermediate values $a = s_a \cdot 2^{e_a}$ (similarly for b, c) and prove that these intermediate values are correctly computed. Then the goal reduces to showing that $|a \cdot b - c| \leq \delta \cdot |a \cdot b|$, which can be easily handled. Nevertheless, checking these three intermediates values could be fairly inefficient especially for large k. This is because the best known method to check exponentiation is to bit decompose each of the exponents: e_a, e_b and e_c, and then use repeated squaring to derive and to prove necessary constraints.

We bypass this overhead by observing that when s_a, s_b and s_c are all normalized, the multiplication between s_a and s_b can stretch the exponents by either $w-1$ or w, thus the exponent e_c is close to $e_a + e_b + w$ up to 1. In other words, $e_c - (e_a + e_b) \in \{w-1, w\}$. With this in mind, let's first transform the inequality into $\left|s_a \cdot s_b - s_c \cdot 2^{e_c-(e_a+e_b)}\right| \leq \delta \cdot |s_a \cdot s_b|$. Then, since $2^{e_c-(e_a+e_b)} \in \{2^{w-1}, 2^w\}$, we can hardcode these two values inside our constraints and add additional constraints which enforces the right value, thus effectively eliminating the need to check otherwise heavy exponentiations.

When it comes to floating number addition, we want to check the following inequality: $|s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b} - s_c \cdot 2^{e_c}| \leq \delta \cdot |s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b}|$.

As in the case of multiplication, we observe a similar relationship between these three exponents. For the sake of simplicity, let's assume $s_a, s_b \geq 0$ and $e_a \geq e_b$. If we add the two floating point numbers: $s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b}$, the exponent of the result of such addition should be either e_a or $e_a + 1$, depending on whether the value $s_a + s_b \cdot 2^{e_b - e_a}$ overfloats (e.g. its value exceeds 2^w) or not. In other words, in this case $e_c - e_a \in \{0, 1\}$. Thus, divide 2^{e_a} from both side of the inequality, we have: $|s_a + s_b \cdot 2^{e_b-e_a} - s_c \cdot 2^{e_c-e_a}| \leq \delta \cdot |s_a + s_b \cdot 2^{e_b-e_a}|$.

Now $s_c \cdot 2^{e_c-e_a} \in \{s_c, 2s_c\}$ and we can add certain constraints to enforce it to take the correct value. Thus we only need to define intermediate values $b = s_b \cdot 2^{e_b-e_a}$ and ask the prover to prove the correctness of such exponentiation. In summary, we only require one bit decomposition of $e_b - e_a$ so as to apply our range proof as in the case of fixed point addition.

Zero-Knowledge. Our protocol can achieve zero-knowledge (ZK) as follows: we instantiate the underlying commit-and-prove proof system with one that achieves ZK. Further, to hide v_i's and prove that they are small, we use the following bit-decomposition method, instead of sending v_j's in clear to the verifier. Specifically, to prove $v_j's$ are at most r-bits, we can we decompose v_j's to bits

$$v_j = s \cdot \sum_{k=0}^{r} v_j[k] \cdot 2^k, \quad v_j[k] = 0 \text{ or } 1$$

and incorporate the above equation and the constraints $v_j[k] \cdot (1 - v_j[k]) = 0, s_j^2 = 1$ into the R1CS. Since there is only a small number of v_j's, the bit-decomposition is dominated by the main body of the protocol, and hence it doesn't affect the efficiency. In this way, ZK follows from the ZK of the underlying commit-and-prove proof system and the hiding property of commitment scheme. For more details, See Section 6.4.

Sub-linear Verification. Our protocol so far needs linear verification time. The bottleneck is that the underlying "commit-and-prove" verification needs to at least read the random linear coefficients $\{r_i\}_i$ in order to verify that $\{v_j\}_j$'s are correctly computed. Since for each gate we write some Equation (1), if we denote $|C|$ as the circuit size, then there are $O(|C|)$ of r_i's, and hence the verification time is linear.

To achieve sub-linear verification, we sample $\mathbf{r} = \{r_i\}_{i \in [n]}$ in a succinct way as follows. We first sample $\mathbf{s} = (s_1, s_2, \ldots, s_{\sqrt{n}})$ and $\mathbf{t} = (t_1, \ldots, t_{\sqrt{n}})$ from a small range, and then generate $\mathbf{r} = \mathbf{s} \otimes \mathbf{t}$. Then we have the prover compute the random linear combination w.r.t $\mathbf{r}$ in the same way as before, but use the underlying "commit-and-prove" to further ensure that $\mathbf{r} = \mathbf{s} \otimes \mathbf{t}$ is computed correctly. Now the verification of the underlying "commit-and-prove" only needs to read $\mathbf{s}$ and $\mathbf{t}$ to verify the proof, and hence the additional verification time becomes sub-linear.

To prove soundness of our sub-linear verification protocol, we use a pigeonhole argument to first extract the values in the "fraction" form. Then we set the modulo to be large enough as before. For more details, see Section 6.3.

4 PRELIMINARIES

We defer definitions of commitment schemes, interactive proof systems and R1CS to the full version.

Sum of Three Squares. Legendre's three-square theorem states that every natural number k that is not of the form $k = 4^a(8b + 7)$, $(a, b \in \mathbb{N})$ can be represented as a sum of three integer squares $(k = x^2 + y^2 + z^2)$. Therefore, for any $k \in \mathbb{N} \setminus \{0\}$, $4k - 3$ can always be written as a sum of three integer squares. Let $n = \log k$ be the bit length of k.

In [27], the authors give an efficient randomized algorithm for identifying (x, y, z) such that $4k - 3 = x^2 + y^2 + z^2$, for any natural number k. The expected running time of the algorithm is $O(n^2)$. The high level idea is as follows: keep guessing an even value x (as one of the squares) until $p = 4k - 3 - x^2$ is a prime congruent to 1 mod 4. If so, compute one square root of the form $\alpha^2 \equiv -1 \bmod p$ and then compute the gcd $y + zi = (\alpha + i,\ p)$ over Gaussian integers. It then follows that $p = y^2 + z^2$ so that $4k - 3 = x^2 + y^2 + z^2$.

5 DEFINITIONS

5.1 Approximate Circuit Computation

Fixed and Floating Point Numbers. For a positive integer w, a *fixed point number* of precision w is a number of the form $s \cdot 2^{-w}$, where the *significand* $s \in [-2^{w-1}, 2^{w-1})$ is an integer represented in w bits.

A *floating point number* of precision w is a pair of integers (s, e), which represent a real number $s \cdot 2^{e-w}$, where $|s| \in [2^{w-1}, 2^w)$ is an integer represented in $w + 1$ bits consisting of 1 sign bit and w bit fraction (called the significand) whose most significant bit is always fixed to 1.

A floating point addition (resp. subtraction, multiplication) gate takes as input two floating point numbers, and outputs a floating point number. A floating point circuit is a circuit where each gate is either a floating point addition, subtraction or multiplication gate.

Definition 5.1 (δ-Approximate Correctness). Let f be the (precise) addition (resp., subtraction, multiplication) function, and let (a, b) be the input wires and c be the output wire of a floating/fixed point addition (resp., subtraction, multiplication) gate.

We say such a floating/fixed point addition (resp. subtraction, multiplication) gate computation is δ-approximately correct, if the relative error is bounded by δ, i.e. $|c - f(a, b)| < \delta |f(a, b)|$.

Definition 5.2 (Floating/Fixed δ-Satisfiable). We say a floating/fixed point circuit C is floating/fixed δ-satisfiable, if there exists an assignment of wires with floating/fixed point numbers such that each gate is δ-approximately correct.

Definition 5.3 (Real δ-Satisfiable). We say a floating point circuit C is real δ-satisfiable, if there exists an assignment of wires with real numbers such that each gate is δ-approximately correct. It is easy to see that a floating δ-satisfiable circuit C is also real δ-satisfiable.

Definition 5.4 (Promise Language for δ-circuit satisfiability). For any $\delta \in (0, 1)$, we define the floating δ-circuit satisfiability problem as the following promise language $(L_\delta, \bar{L_\delta})$:

- An (circuit) instance $C \in L_\delta$ if C is floating δ-satisfiable. In this case there exists witness $\mathbb{W}$ corresponding to the wire assignments, where each coordinate of $\mathbb{W}$ is a floating point number of w-precision.
- An (circuit) instance $C \notin \bar{L_\delta}$ if C is not real δ-satisfiable.

Remark 1. *The aforementioned definitions of floating point gates and circuit, floating δ-satisfiability etc. can also be extended to fixed point number computations.*

We remark that here we have a theoretical gap between the completeness and soundness. That is, there could be a circuit that is real δ-satisfiable, but not floating δ-satisfiable, since not every real number can be represented as floating point number. However, we expect such gap to be narrow in practice. As we discussed in Section 2.1, we expect the floating point program used in practice is robust to any small perturbations bounded by the relative error upper bound. Hence, if we round the real number to the nearest floating point number, then the circuit is still satisfiable for those floating point numbers.

5.2 Interactive Proofs for Floating Point Computations

An interactive proofs for floating δ-circuit satisfiability problem $(L_\delta, \bar{L_\delta})$ is a pair of algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$, with the following syntax.

- $\mathcal{G}(1^\lambda)$: The CRS generation algorithm takes as input the security parameter λ, and it outputs a crs.
- $\mathcal{P}(\text{crs}, C, \mathbb{W})$: The prover is given the crs, a circuit $C \in L_\delta$, and a witness $\mathbb{W}$ for C, it outputs a proof π.
- $\mathcal{V}(\text{crs}, C, \pi)$: The verifier takes as input the crs, a circuit C, and the proof π. Then it decides to accept or reject.

Furthermore, we require the following properties.

- **Completeness.** For every circuit instance $C \in L_\delta$ and its witness $\mathbb{W}$, the honest prover's proof π should always be accepted, i.e.

$$\Pr\left[\text{crs} \leftarrow \mathcal{G}(1^\lambda),\ \pi \leftarrow \mathcal{P}(1^\lambda, \text{crs}, C, \mathbb{W}) : \mathcal{V}(1^\lambda, \text{crs}, C, \pi) = 1\right] = 1.$$

- **Soundness.** There exists some negligible function ϵ such that for every circuit instance $C \in \bar{L_\delta}$, and for all probabilistic polynomial time malicious prover $\mathcal{P}^*$, its proof π^* is accepted with probability at most ϵ, i.e.

$$\Pr\left[\text{crs} \leftarrow \mathcal{G}(1^\lambda),\ \pi^* \leftarrow \mathcal{P}^*(1^\lambda, \text{crs}, C, \mathbb{W}) : \mathcal{V}(1^\lambda, \text{crs}, C, \pi^*) = 1\right] \le \epsilon.$$

Succinctness. We say the protocol is succinct, if the size of the proof π is bounded by $|\mathbb{W}|^{o(1)}$, where $|\mathbb{W}|$ is the bit-length of $\mathbb{W}$.

Non-interactive Proofs. We say a proof system is non-interactive, if it only has one round.

Remark 2. *We can transform any public-coin interactive protocols to a non-interactive one using the Fiat-Shamir transformation [17]. The succinctness and the zero-knowledge properties are preserved.*

6 COMMIT-AND-PROVE FOR FLOATING POINT COMPUTATIONS

In this section, we build a commit-and-prove protocol for floating point number computations. Our construction is generic from any commit-and-prove for R1CS with argument of knowledge property.Before going into the technical details, we firstly give a high level overview of our ideas.

Recall that, for floating point computation, around each gate we want to prove constraints of the form $|c - f(a, b)| < \delta |f(a, b)|$, where a, b, c are values on two inputs and one output wire, and $f(a, b)$ is the precise value of the gate output. For simplicity, let's consider f as a multiplication gate in this overview, as other gates can be handled in a similar way. Recall that, a floating point number is represented as a *significand* s and an *exponent* e (Section 5.1). For multiplications, we need to prove that the exponent part c is roughly the addition of the exponent part of a and b, which can be

done easily. Then the only complication left is how to prove the significand parts of c is also approximately correct, which is essentially proving the δ-approximate correctness for fixed point computation. For this reason, let's only consider fixed point computation instead of floating point computation in this section. We will extend the same techniques to floating point numbers in (Section 6.5).

Since a, b, c are fixed point numbers, they can be represented as $a = a' \cdot 2^{-w}, b = b' \cdot 2^{-w}, c = c' \cdot 2^{-w}$, where $a', b', c' \in [-2^{w-1}, 2^{w-1})$ are w-bit integers. For any $\delta \in (0,1)$, it suffices to consider $\delta = \Delta_1/\Delta_2$, $(\Delta_1 < \Delta_2)$, where $\Delta_1, \Delta_2 \in (0, 2^w]$ are both positive integers. Then the constraint we want to prove becomes

$$|\Delta_2 \cdot (2^w c' - a' \cdot b')| < |\Delta_1 \cdot a' \cdot b'|.$$

Now we add two intermediate variables $x := \Delta_2 \cdot (2^w c' - a' \cdot b')$ and $y := \Delta_1 \cdot a' \cdot b'$. Notice that the constraint $|x| < |y|$ is equivalent to $x^2 < y^2$. It thus suffices to prove that $z := (x+y)(y-x) > 0$.

Next, we present our construction.

6.1 Construction

Before we build the commit-and-prove for floating point computations, we list the necessary ingredients as follows.

Ingredients.

- A commitment scheme $(\mathsf{KGen}, \mathsf{Com})$ with hiding and binding property.
- A commit-and-prove protocol $(\mathsf{KGen}, \mathsf{Com}, \mathcal{P}, \mathcal{V})$ for the commitment scheme $(\mathsf{KGen}, \mathsf{Com})$ and R1CS over a finite field $\mathbb{F}_p$ with argument of knowledge property.
- A compiler R1CSCompiler that takes as input a circuit C for floating/fixed point numbers, and outputs a R1CS instance.

Construction. The construction of commit-and-prove for floating/fixed point computation is depicted in Figure 1.

To reduce the concrete size of p, the actual random linear combination in our protocol is in fact more fine-grained than the overview described in Section 3. In particular, in the case of fixed point computation, instead of doing a single random linear combination over all wire values, we partition all the wires into two parts: the wires with $\lesssim 3w$-bits, and the wires with $\gtrsim 3w$-bits but $\lesssim 6w$-bits. Then we use two random linear combinations to test them separately. Hence, we have the R1CSCompiler output two disjoint sets S_1, S_2 to contain the wire values for these two kinds of wires. The case of floating point computation is handled in a similar way.

R1CSCompiler **for Fixed Point Circuits.** We first construct the R1CSCompiler for fixed point computation as follows. Recall that a fixed point number of precision w is a number of the form $s \cdot 2^{-w}$, where $s \in [-2^{w-1}, 2^{w-1})$ is an integer represented in w bits.

- The compiler takes as input a fixed point circuit C and δ. It parses $\delta = \Delta_1/\Delta_2$, where Δ_1, Δ_2 are both integers of w-bit, and initializes two empty sets $S_1 = \emptyset, S_2 = \emptyset$, and an empty R1CS instance $\mathbb{X}$.
- For each gate g_i in C, let $a_i \cdot 2^{-w}, b_i \cdot 2^{-w}$ be the input wires, and $c_i \cdot 2^{-w}$ be the output wire.
  - If g_i is a multiplication gate, compute $x_i = \Delta_2(2^w c_i - a_i \cdot b_i)$, $y_i = \Delta_1 \cdot a_i \cdot b_i$ as follows. Let $\bar{g}_i := a_i \cdot b_i, \quad x_i := 2^w \Delta_2 \cdot c_i - \Delta_2 \cdot \bar{g}_i, \quad y_i := \Delta_1 \cdot \bar{g}_i$.

Commit-and-prove for Floating/Fixed Point Computation

(1) The prover and the verifier run R1CSCompiler to obtain a R1CS instance,
$$\left(\mathbb{X}_{\mathsf{R1CS}}, \{S_i, w_i\}_{i\in[\ell]}\right) \leftarrow \mathsf{R1CSCompiler}(1^\lambda, C, \delta),$$
where $\mathbb{X}_{\mathsf{R1CS}}$ is a R1CS instance with n variables. The prover can also derive the corresponding R1CS witness $\mathbb{W}_{\mathsf{R1CS}}$ from a witness $\mathbb{W}$ of C. The prover generates $c := \mathsf{Com}(\mathbb{W}_{\mathsf{R1CS}}; u)$ with some randomness u, and sends c to the verifier.
(2) The verifier sends a series of random coefficients $\mathbf{r} \leftarrow [0, 2^\kappa)^n$ to the prover.
(3) The prover sends back $\{v_i\}_{i\in[\ell]}$, where $v_i = \left\langle \mathbb{W}_{\mathsf{R1CS}}|_{S_i}, \mathbf{r}|_{S_i} \right\rangle \in \mathbb{F}_p$ ($\mathbf{r}|_{S_i} \in \mathbb{F}_p^n$ agrees with $\mathbf{r}$ for all entries in S_i, and is 0 otherwise).
(4) The prover and the verifier augment the R1CS instance $\mathbb{X}_{\mathsf{R1CS}}$ by appending the constraint $\left\langle \mathbb{W}_{\mathsf{R1CS}}|_{S_i}, \mathbf{r}|_{S_i} \right\rangle = v_i$, for all $i \in [\ell]$. They denote $\mathbb{X}'_{\mathsf{R1CS}}$ as the augmented R1CS instance. Then they execute the commit-and-prove protocol.
$$\mathcal{P}\left(1^\lambda, (\mathbb{X}'_{\mathsf{R1CS}}, u)\right) \leftrightarrow \mathcal{V}(1^\lambda, \mathbb{X}'_{\mathsf{R1CS}}).$$
The verifier checks that $v_i \in [-n \cdot 2^{w_i+\kappa}, n \cdot 2^{w_i+\kappa})$ for each $i \in [\ell]$. It accepts if both the range check passes and $\mathcal{V}$ accepts. Otherwise the verifier rejects.

Figure 1: Description of commit-and-prove for floating/fixed point computation.

  - Otherwise if g_i is an addition/subtraction gate, compute $x_i = \Delta_2(a_i \pm b_i - c), y_i = \Delta_1(a_i \pm b_i)$ as follows. Let $\bar{g}_i := a_i \pm b_i, \quad x_i := \Delta_2 \cdot \bar{g}_i - \Delta_2 \cdot c_i, \quad y_i := \Delta_1 \cdot \bar{g}_i$.
  - Compute $z_i = (x_i + y_i) \cdot (y_i - x_i)$ as follows. Let $z_i^+ := x_i + y_i, \quad z_i^- := y_i - x_i, \quad z_i := z_i^+ \cdot z_i^-$.
  - Add new variables $\gamma_{i,1}, \gamma_{i,2}, \gamma_{i,3}$, and verify $4z_i - 3 = \gamma_{i,1}^2 + \gamma_{i,2}^2 + \gamma_{i,3}^2$ as follows. Let $u_{i,k} := \gamma_{i,k}^2, \forall k \in [3], \mathsf{sum}_i := u_{i,1} + u_{i,2}, 4z_i - 3 := \mathsf{sum}_i + u_{i,3}$.
  - Add all above constraints to the R1CS instance $\mathbb{X}$, and update S_1 and S_2 as follows,
$$S_1 := S_1 \cup \{a_i, b_i, c_i, \bar{g}_i, x_i, y_i, z_i^+, z_i^-, \{\gamma_{i,k}\}_{k\in[3]}\},$$
$$S_2 := S_2 \cup \{z_i, \{u_{i,k}\}_{k\in[3]}, \mathsf{sum}_i\}.$$
- Finally, output $(\mathbb{X}, \{(S_1, w_1 := 3w+2), (S_2, w_2 := 6w+4)\})$.

6.2 Security Proofs

We defer the security proof to the full version.

6.3 Achieving Sublinear Verification

We defer how to achieve sublinear verification to the full version.

6.4 Achieving Zero-knowledge

Our protocol can be easily modified to achieve zero knowledge property. We refer the reader to Section 3 for a high-level overview. For this part, we are going to assume that the underlying commitment scheme in the commit-and-prove system to be *additively*

homomorphic: $\mathsf{Com}(x; r_1) + \mathsf{Com}(y; r_2) = \mathsf{Com}(x + y, r_1 + r_2)$ (see the full version for its definition.) This is in fact already the case for many existing commit-and-prove systems. We achieve zero knowledge via the following simple modifications:

(1) In step 1 of Figure 1, instead of committing to just $\mathbb{W}_{\mathsf{R1CS}}$, the prover commits to the concatenation of $\mathbb{W}_{\mathsf{R1CS}}||0^m$ as $c := \mathsf{Com}(\mathbb{W}_{\mathsf{R1CS}}||0^m; u)$, where $m = \sum_{i\in[l]} (\log(n) \cdot (w_i + \kappa) + 1)$ is the total bit length of all $v_{i\in[l]}$'s. It then sends c to the verifier.
(2) In step 3 of Figure 1, the prover still defines $\{v_i\}_{i\in[l]}$ accordingly, however not sending them to the verifier. It instead computes the bit decomposition of for each v_i as $v_i = s_i \cdot \sum_{k=0}^{\log(n)\cdot(w_i+\kappa)} v_i[j] \cdot 2^k$, where $v_i[j]$ is the j-th bit of v_i and $s_i \in \{-1, +1\}$ is the sign of v_i. It then sets $\mathbf{b}_i := s_i||\{v_i[j]\}$, and then sends the commitment $c' := \mathsf{Com}(0^{|\mathbb{W}_{\mathsf{R1CS}}|}||\{\mathbf{b}_i\}_{i\in[l]}; u')$ to the verifier.
(3) (Adding Range Constraints) In step 4, the prover and verifier add the constraints that $v_i \in [-n \cdot 2^{w_i+\kappa}, n \cdot 2^{w_i+\kappa})$ for each $i \in [\ell]$ to the R1CS instance. These range constraints can be enforced via the same bit-decomposition method as follows:
   – For each $i \in [\ell]$, add constraint $v_i = s_i \cdot \sum_{k=0}^{\log(n)\cdot(w_i+\kappa)} v_i[j] \cdot 2^k$.
   – Add constraints that $v_i[j] \cdot (1 - v_i[j]) = 0, s_i^2 = 1$.
(4) (Format Checking) In step 4, the prover and verifier define another R1CS instance $\mathbb{X}^*_{\mathsf{R1CS}}$ checking that the witness committed in c' is well-formed: it is correctly padded with the zero vector $0^{|\mathbb{W}_{\mathsf{R1CS}}|}$. This check ensures that the witness $\mathbb{W}_{\mathsf{R1CS}}$ committed in c remains unchanged.
(5) In step 4, the prover and verifier initiate the underlying commit-and-prove protocol with zero-knowledge property, and then proceed checking both R1CS instances $\mathbb{X}'_{\mathsf{R1CS}}$ and $\mathbb{X}^*_{\mathsf{R1CS}}$, with respect to the commitments $c + c'$ and c'.

We defer the proof of zero knowledge to the full version. At a high level the zero-knowledge property can be proven as follows. First, we use the zero-knowledge simulator of the underlying commit-and-prove to simulate the transcript of commit-and-prove in step 4. Then we use the hiding property of the underlying commitment scheme to argue that the commitments c, c' sent in step 1 and 3 can be simulated.

To argue soundness, notice that due to the range constraints, each v_i must be the correct range. Furthermore, the witness $\mathbb{W}_{\mathsf{R1CS}}$ committed in the first commitment c cannot be modified by the prover since we also enforce that the value committed in c' must start with all 0's. We defer the soundness proof to the full version.

Finally, via the Fiat-Shamir transformation, we obtain a ZK-SNARG for floating/fixed point computations.

6.5 Extension to Floating Point Computation

Recall that a *floating point number* of precision w is a pair of integers (s, e), which represent a real number $s \cdot 2^{e-w}$, where $|s| \in [2^{w-1}, 2^w)$ is an integer represented in $w + 1$ bits with 1 sign bit and w bit fraction (called the significand) where the fraction is always normalized such that the most significant bit is fixed to 1. We first present a high-level overview of our R1CS compiler which compiles any floating point computation into a R1CS instance. Starting from this point, we make two inherent relaxations for the ease of our compiler construction. Firstly, we fix $\delta = 2^{-w}$ for addition/subtraction gate and $\delta = 2^{-2w}$ for multiplication gate. Secondly, we relax all the constraints (as defined in 5.1) from strict inequalities to inequalities.

Adding/Subtracting Two Floating Point Numbers. Suppose we want to add/subtract two floating point numbers: $s_a \cdot 2^{e_a-w}$ and $s_b \cdot 2^{e_b-w}$. Let the outcome of such addition/subtraction be $s_c \cdot 2^{e_c-w}$. Following δ-approximate correctness, we want to ensure that:

$$\left|s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b} - s_c \cdot 2^{e_c}\right| \le \delta \cdot \left|s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b}\right|,$$

where we have multiplied both sides by 2^w to simplify the inequality. We also incorporate subtraction into addition by allowing each summand to take negative values.

A Naive Approach. Checking floating point numbers addition is particularly challenging due to the fact that we also need to take care of their exponents. To see why, consider a naive (however very inefficient) way to convert this inequality into R1CS constraints: Let's first define intermediate variables and constraints that mimic all exponentiations: (e.g. we can define $\lambda_a := 2^{e_a}$ and so on). Once we have these intermediate exponentiations, one can then check:

$$|s_a \cdot \lambda_a + s_b \cdot \lambda_b - s_c \cdot \lambda_c| \le \delta \cdot |s_a \cdot \lambda_a + s_b \cdot \lambda_b|.$$

If we take the upper bound of the size of each summand, we can essentially view each summand as a (very large) fixed-point number. In this way we can apply our previous range proof technique to this inequality just like the case of fixed point addition.

This naive approach mainly suffers from two sources of inefficiency: Firstly, the best known method to check exponentiation requires bit decomposing the exponent, and then use repeated squaring algorithm to break down the exponentiation procedure into one multiplication and addition at each step, thus turning them into constraints compatible with R1CS. Clearly, we need to bit decompose all of e_a, e_b and e_c, turning them into roughly $3 \times (2k + 1)$ constraints with $3 \times (2k)$ new variables. We reduce the number of constraints and variables by roughly a factor of 3, via only one bit decomposition and some extra constraints. For large value of k (for example, $k = 11$ in a 64-bit floating point number) this will greatly increase the prover time. Secondly, for large k, the value $\lambda_a = 2^{e_a} \approx 2^{2^k}$ will typically become very large. This will cause the upper bound on each summand to significantly grow, which pushes the choice of the modulus p to become large (for example, in a 32-bit floating point number, $2^k < 128$ and we need to choose p to be around 670 bits). In our approach, we reduce this upper bound so that it no longer depends on k, but on $\log(w)$ instead. This will allow us to choose a smaller p to again reduce prover computation (in the last example, $\log(w) \approx 5$ so we can choose p to be around 270 bits).

Converting Addition into Constraints: Our Approach. We begin explaining our idea with a simple yet useful observation:

Claim 1. *For $|s_a|, |s_b|, |s_c| \in \left[2^{w-1}, 2^w\right)$ and $\delta = 2^{-w}$, if $|s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b} - s_c \cdot 2^{e_c}| \le \delta \cdot |s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b}|$, then $\min(e_a - e_b, e_b - e_c, e_a - e_c) \in \{-1, 0, 1\}$.*

Proof. First suppose that among s_a, s_b and s_c, two of them have the same signs and the other has the opposite sign, we claim that this

is impossible: by renaming the variables, we can always assume that s_a and s_b have the same signs. Furthermore, since we can always negate the signs of all variables within an absolute clause, let's always assume that $s_a, s_b \geq 0$, such that $-s_c \geq 0$. Therefore,

$$\begin{aligned}&\left|s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b} - s_c \cdot 2^{e_c}\right| \\ =&s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b} - s_c \cdot 2^{e_c} \\ \geq&s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b}.\end{aligned}$$

Since $\delta \cdot s_a < s_a$ and $\delta \cdot s_b < s_b$, $s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b} > \delta \cdot |s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b}|$, which is a contradiction. Now suppose that all of s_a, s_b and s_c have the same (positive) signs, and furthermore assume $s_a \cdot 2^{e_a} \geq s_b \cdot 2^{e_b}$ by renaming the variables if necessary, then we have: $|s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b} - s_c \cdot 2^{e_c}| \leq \delta \cdot (s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b})$,

$$(1-\delta)s_a \cdot 2^{e_a} + (1-\delta)s_b \cdot 2^{e_b} \leq s_c \cdot 2^{e_c} \leq (1+\delta)s_a \cdot 2^{e_a} + (1+\delta)s_b \cdot 2^{e_b}.$$

Since $\delta = 2^{-w}$, and $s_a \geq 2^{w-1}$, $(1-\delta)s_a \approx s_a \approx (1+\delta)s_a$ and $(1-\delta)s_b \approx s_b \approx (1+\delta)s_b$, so that we have: $s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b} \approx s_c \cdot 2^{e_c}$. Since $s_a \cdot 2^{e_a} \geq s_b \cdot 2^{e_b}$, then: $s_c \cdot 2^{e_c} \approx s_a \cdot 2^{e_a} + s_b \cdot 2^{e_b} \leq s_a \cdot 2^{e_a+1}$ Given that both s_a, s_c are normalized, by comparing the terms, we get $e_c \leq e_a + 1$, hence $\min(e_a - e_b, e_b - e_c, e_a - e_c) \in \{-1, 0, 1\}$. □

To express each floating point number addition into a number of R1CS constraints, let's utilize the above observation by dividing 2^{e_c} from both sides of the inequality and also defining $\theta_a := e_a - e_c$ and $\theta_b := e_b - e_c$. Notice that we have three possible scenarios:

(1) **Case I:** $\theta_a \in \{-1, 0, 1\} \Leftrightarrow e_a - e_c \in \{-1, 0, 1\}$.
(2) **Case II:** $\theta_b \in \{-1, 0, 1\} \Leftrightarrow e_b - e_c \in \{-1, 0, 1\}$.
(3) **Case III:** $\theta_a - \theta_b \in \{-1, 0, 1\} \Leftrightarrow e_a - e_b \in \{-1, 0, 1\}$.

Meanwhile, the inequality translates into: $\left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta_b} - s_c\right| \leq \delta \cdot \left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta_b}\right|$.

Our goal is to use only one bit decomposition to enforce two constraints: ($\lambda_a = 2^{\theta_a}$ and $\lambda_b = 2^{\theta_b}$). In order to achieve this goal, observe that in each of the three possible scenarios, we only need to enforce one such constraint (hence one bit decomposition): $\lambda_\theta = 2^\theta$, where θ is either θ_a or θ_b. The reason is as follows:

(1) **Case I:** Since θ_a is small, we can rewrite its constraint using Lagrange interpolation as: $\lambda_{\theta_a} = \frac{\theta_a(\theta_a-1)}{4} + \theta_a(\theta_a+1) - (\theta_a+1)(\theta_a-1)$. Now set $\theta = \theta_b$, and add the additional constraint that $\lambda_\theta = 2^\theta$.
(2) **Case II:** This case resembles a similar treatment as Case I, where $\theta = \theta_a$.
(3) **Case III:** Let's set $\theta = \theta_b$, and add the constraint that $\lambda_\theta = 2^\theta$. Now define $\Delta_{a,b} = \theta_a - \theta_b$, which is again small, meaning that we can again use Lagrange interpolation to derive the constraint for θ_a as:

$$\lambda^*_{\theta_a} = \frac{\Delta_{a,b}(\Delta_{a,b}-1)}{4} \cdot \lambda_\theta + \Delta_{a,b}(\Delta_{a,b}+1) \cdot \lambda_\theta - (\Delta_{a,b}+1)(\Delta_{a,b}-1) \cdot \lambda_\theta.$$

In order to determine whether $\theta = \theta_a$ or $\theta = \theta_b$, we use three relaxed indicator variables and enforce each of their values to take either 0 or 1:

(1) $\mathbb{1}_{\theta_a}(1-\mathbb{1}_{\theta_a}) = 0$, and $\mathbb{1}_{\theta_a} = 1 \implies$ Case I happens (details later);
(2) $\mathbb{1}_{\theta_b}(1-\mathbb{1}_{\theta_b}) = 0$, and $\mathbb{1}_{\theta_b} = 1 \implies$ Case II happens;
(3) $\mathbb{1}_{\Delta_{a,b}}(1-\mathbb{1}_{\Delta_{a,b}}) = 0$, and $\mathbb{1}_{\Delta_{a,b}} = 1 \implies$ Case III happens.

Notice that we do not require the reverse of these statements to be true. Now we add two more constraints:

(1) $\mathbb{1}_{\theta_a} + \mathbb{1}_{\theta_b} + \mathbb{1}_{\Delta_{a,b}} = 1$;
(2) $\theta = \mathbb{1}_{\theta_a} \cdot \theta_b + \mathbb{1}_{\theta_b} \cdot \theta_a + \mathbb{1}_{\Delta_{a,b}} \cdot \theta_b$.

These constraints will enforce θ to take the value of either θ_a or θ_b, depending on the desired scenario.

Turn Inequality into Equality for Large θ. A subtle issue in the above construction is that if the absolute value of θ becomes large, the modulus p needs also to grow large for soundness to hold. To understand this issue, let's take case I as an example: we want to use range proof to show the following inequality:

$$\left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta} - s_c\right| \leq \delta \cdot \left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta}\right|.$$

Notice that both θ_a and θ may be negative (thus leaving fractional numbers on both sides), and we are checking this inequality over mod p. To ensure the inequality also holds over the reals, we return to our previous techniques: First let's multiplying both sides by $2^{|\theta|+1}$ to ensure that all of the exponents are positive (thus removing fractional numbers):

$$\begin{aligned}&\left|s_a \cdot 2^{\theta_a+|\theta|+1} + s_b \cdot 2^{\theta+|\theta|+1} - s_c \cdot 2^{|\theta|+1}\right| \\ \leq&\delta \cdot \left|s_a \cdot 2^{\theta_a+|\theta|+1} + s_b \cdot 2^{\theta+|\theta|+1}\right|.\end{aligned}$$

Now if we choose the bit length of modulus p to be larger than $2\log(n)+4w+3\kappa+12+2^2 \cdot |\theta|$, the above inequality will hold over the reals. Clearly, the bit length of p needs to grow with $2^{2|\theta|} \approx 2^{k+1}$.

The increased group size will induce a significant cost on prover computation. Fortunately, we observe that when $|\theta|$ exceeds $w+2$, we can get rid of the dependence on $|\theta|$ by substituting all inequalities with different equalities for each of the aforementioned three scenarios:

(1) **Case I:** There are two possibilities:
   (a) $\theta > w+2$: Since s_a, s_b, s_c are normalized and $|\theta_a| \leq 1$, we can bound $\left|s_a \cdot 2^{\theta_a} - s_c\right| < 2^2 \cdot |s_b|$. Therefore, $\left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta} - s_c\right| > |s_b \cdot 2^w| > \delta \cdot \left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta}\right|$. Thus we always reject.
   (b) $\theta < -(w+2)$: In this case $\left|s_b \cdot 2^{\theta}\right|$ is relatively small compared to $\left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta}\right|$. Our intuition is that one can simply drop the summand $\left|s_b \cdot 2^{\theta}\right|$ while still obeying the inequality. More formally, since $\left|s_b \cdot 2^{\theta}\right| < \left|s_b \cdot 2^{-w-2}\right| < \left|s_a \cdot 2^{-w-1}\right| < \delta \cdot \left|s_a \cdot 2^{\theta_a}\right| \leq \delta \cdot \left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta}\right|$, we can instead enforce the equality constraint that $s_a \cdot 2^{\theta_a} = s_c$. This is done by checking $s_a = s_c$ and $e_a = e_c$.
(2) **Case II:** This case resembles the analysis of Case I.
(3) **Case III:** There are two possibilities:
   (a) $s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta} = 0$: In this case, we simply check if $s_c = 0$ (In our construction we ignore this case for simplicity).
   (b) Otherwise, first assume $\theta > w+2$, notice that $\left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta}\right| \geq 2^{\theta} |s_b + s_a/2| \geq 2^{\theta-1}$. Furthermore, $\left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta}\right| \leq 2^{\theta} |s_b + 2s_a| \leq 2^{\theta+2}$. As a result, we

deduce that:

$$\begin{aligned}&\left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta} - s_c\right| \\ \geq &\left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta}\right| - |s_c| \\ \geq &2^{\theta-1} - |s_c| \geq 2^{\theta-1} - 2^w,\end{aligned}$$

on the other hand, $\delta \cdot \left|s_a \cdot 2^{\theta_a} + s_b \cdot 2^{\theta}\right| \leq 2^{\theta+2-w}$.

For $w > 2$ and $\theta > w + 2$, one can observe that $2^{\theta+2-w} < 2^{\theta-1} - 2^w$, thus the desired inequality will not hold. The other scenario where $\theta < -(w + 2)$ is very similar. Therefore we should always reject when $|\theta| > w + 2$.

To conclude, we can choose the bit length of p to grow with $2^2 \cdot \max(\theta, w + 2)$ instead of θ. In practice we will simplify our constraints by picking the cutoff to be $w + 8 = 2^5$ for 32 bit floating point numbers, or $w + 9 = 2^6$ for 64 bit floating point numbers.

Multiplying Two Floating Point Numbers. Suppose we want to multiply two floating point numbers: $s_a \cdot 2^{e_a - w}$ and $s_b \cdot 2^{e_b - w}$. Let the outcome of such multiplication be $s_c \cdot 2^{e_c - w}$. Following δ-approximate correctness 5.1, we want to ensure that:

$$\left|s_a \cdot 2^{e_a} \cdot s_b \cdot 2^{e_b} - s_c \cdot 2^{e_c}\right| \leq \delta \cdot \left|s_a \cdot 2^{e_a} \cdot s_b \cdot 2^{e_b}\right|.$$

Our methodology for multiplications will align with our previous route for addition. We begin with yet another simpler observation about the relationship between these exponents (e_a, e_b, e_c).

Claim 2. *For $|s_a|, |s_b|, |s_c| \in \left[2^{w-1}, 2^w\right)$ and $\delta = 2^{-w}$, if $|s_a \cdot 2^{e_a} \cdot s_b \cdot 2^{e_b} - s_c \cdot 2^{e_c}| \leq \delta \cdot |s_a \cdot 2^{e_a} \cdot s_b \cdot 2^{e_b}|$, then $e_c - (e_a + e_b) \in \{w - 1, w\}$.*

Proof. First, let's rewrite the inequality as: $\left|s_a \cdot s_b - s_c \cdot 2^{e_c - (e_a + e_b)}\right| \leq \delta \cdot |s_a \cdot s_b|$. Since $|s_a|, |s_b| \in \left[2^{w-1}, 2^w\right)$, we have $2^{2w-2} \leq |s_a \cdot s_b| < 2^{2w}$. WLOG assume $s_a s_b \geq 0$ so that $s_c \geq 0$ (the other case is similar), then: $s_a s_b (1 - \delta) \leq s_c \cdot 2^{e_c - (e_a + e_b)} \leq s_a s_b (1 + \delta)$.

Since $\delta = 2^{-2w}$, $s_a s_b (1 - \delta) \approx s_a s_b \approx s_a s_b (1 + \delta)$, thus we can derive the following lower and upper bound: $2^{2w-2} \leq s_c \cdot 2^{e_c - (e_a + e_b)} < 2^{2w}$. Since $s_c \in [2^{w-1}, 2^w)$, by comparing the terms, we deduce that $e_c - (e_a + e_b) \in \{w - 1, w\}$. □

Converting Multiplication into Constraints. Let's define $\theta = e_c - (e_a + e_b)$. Applying the previous claim, if $\theta \notin \{w - 1, w\}$, we simply reject (unless $s_a \cdot s_b = 0$, which we ignore for simplicity). If $\theta \in \{w - 1, w\}$, we can compute the exponentiation $\lambda_\theta := 2^\theta$ using Lagrange Interpolation:

$$\lambda_\theta = (\theta - w + 1) \cdot 2^w - (\theta - w) \cdot 2^{w-1}.$$

Then we can apply range proof as before. In the next part, we present our compiler which converts floating point number computation into R1CS constraints.

R1CSCompiler **for Floating Point Computation.**

- The compiler takes as input a floating point circuit C and $\delta \in \{2^{-w}, 2^{-2w}\}$ and initializes an empty R1CS instance $\mathbb{X}$. It also initializes empty sets S_1 and S_2.
- For each gate g in C, let $s_a \cdot 2^{e_a - w}$, $s_b \cdot 2^{e_b - w}$ be the input wire values, $s_c \cdot 2^{e_c - w}$ be the output wire value. Let k be the bit length of exponents and let $(\theta[1], ..., \theta[k])$ be the natural bit decomposition of θ such that $\theta = \sum_{i=1}^k 2^i \cdot \theta[i]$.
- For every addition/subtraction gate g, we introduce the following new variables and constraints:

Constraints for Indicators	Notes
$\theta_a = e_a - e_c$; $\theta_b = e_b - e_c$; $\Delta_{a,b} = e_a - e_b$; $\mathbb{1}_{\theta_a}(1 - \mathbb{1}_{\theta_a}) = 0$; $\mathbb{1}_{\theta_b}(1 - \mathbb{1}_{\theta_b}) = 0$; $\mathbb{1}_{\Delta_{a,b}}(1 - \mathbb{1}_{\Delta_{a,b}}) = 0$.	Create indicators for three different cases.
$\mathbb{1}_{\theta_a} \cdot (\theta_a + 1)\theta_a(\theta_a - 1) = 0$; $\mathbb{1}_{\theta_b} \cdot (\theta_b + 1)\theta_b(\theta_b - 1) = 0$; $\mathbb{1}_{\Delta_{a,b}} \cdot (\Delta_{a,b}+1)\Delta_{a,b}(\Delta_{a,b}-1) = 0$.	Defining each indicator.
$\mathbb{1}_{\theta_a} + \mathbb{1}_{\theta_b} + \mathbb{1}_{\Delta_{a,b}} = 1$; $\theta = \mathbb{1}_{\theta_a} \cdot \theta_b + \mathbb{1}_{\theta_b} \cdot \theta_a + \mathbb{1}_{\Delta_{a,b}} \cdot \theta_b$.	Decide the value θ to be bit decomposed.

Constraints for Bit Decompositions	Notes
$\forall i \in [k] : \theta[i] \cdot (1 - \theta[i]) = 0$; $(1 - \text{sgn}\theta) \cdot (1 + \text{sgn}\theta) = 0$.	Create k bits and a sign variable.
$\theta = \text{sgn}\theta \cdot \sum_{i=1}^k 2^i \cdot \theta[i]$;	Bit decomposition of θ.
$\mathbb{1}_{\theta>0} = \frac{\text{sgn}\theta+1}{2}$;	Defining indicator for $\theta > 0$.
$\mathbb{1}_{\|\theta\| \leq w+2} = \prod_{i=\log(w+2)+1}^k (1 - \theta[i])$.	Defining indicator for $\|\theta\| \leq w + 2$.

Constraints for Exponentiations	Notes
$\lambda_{\theta,1} = 1$; $\forall i \in [\log(w + 2)] : \lambda_{\theta,i+1} = (1 - \theta[i]) \cdot \lambda_{\theta,i} + \theta[i] \cdot \lambda_{\theta,i} \cdot 2^{2^i}$; $\lambda_\theta = \lambda_{\theta,\log(w+2)}$.	Truncate θ to $\log(w + 2)$ bits and then define $\lambda_\theta = 2^\theta$ using repeated squares.
$\forall \alpha \in \{a, b\} : \lambda_{\theta_\alpha} = \frac{\theta_\alpha(\theta_\alpha-1)}{4} + \theta_\alpha(\theta_\alpha + 1) - (\theta_\alpha + 1)(\theta_\alpha - 1)$; $\lambda^*_{\theta_a} = \frac{\Delta_{a,b}(\Delta_{a,b}-1)}{4} \cdot \lambda_\theta + \Delta_{a,b}(\Delta_{a,b} + 1) \cdot \lambda_\theta - (\Delta_{a,b} + 1)(\Delta_{a,b} - 1) \cdot \lambda_\theta$.	$\lambda_{\theta_\alpha} = 2^{\theta_\alpha}$ whenever $\theta_\alpha \in \{-1, 0, 1\}$. $\lambda^*_{\theta_a} = 2^{\theta_a}$ whenever $\Delta_{a,b} \in \{-1, 0, 1\}$.

Constraints for Range Proofs with Small θ	Notes
$x_I = \delta^{-1} \cdot (s_a \cdot \lambda_{\theta_a} + s_b \cdot \lambda_\theta - s_c)$, $y_I = s_a \cdot \lambda_{\theta_a} + s_b \cdot \lambda_\theta$; $x_{II} = \delta^{-1} \cdot (s_a \cdot \lambda_\theta + s_b \cdot \lambda_{\theta_b} - s_c)$, $y_{II} = s_a \cdot \lambda_\theta + s_b \cdot \lambda_{\theta_b}$; $x_{III} = \delta^{-1} \cdot (s_a \cdot \lambda^*_{\theta_a} + s_b \cdot \lambda_\theta - s_c)$, $y_{III} = s_a \cdot \lambda^*_{\theta_a} + s_b \cdot \lambda_\theta$; $\forall j \in \{\mathrm{I}, \mathrm{II}, \mathrm{III}\}$, $\Theta_j = (x_j + y_j)(y_j - x_j)$;	We first assign to variable Θ_j the inequality to be checked (e.g. $(x_j + y_j)(y_j - x_j) > 0$) for each of three cases.
$z = \mathbb{1}_{\lvert\theta\rvert \le w+2} \cdot (\mathbb{1}_{\theta_a} \cdot \Theta_I + \mathbb{1}_{\theta_b} \cdot \Theta_{II} + \mathbb{1}_{\Delta_{a,b}} \cdot \Theta_{III})$; $4z - 3 = \gamma_1^2 + \gamma_2^2 + \gamma_3^2$.	Using the indicators, the variable z will correspond to the desired case. Apply range proof (we omit intermediate steps).

Constraints for Large θ	Notes
$(1 - \mathbb{1}_{\lvert\theta\rvert \le w+2}) \cdot \mathbb{1}_{\theta > 0} \cdot \mathbb{1}_{\theta_a} = 0$;	In case I, reject if $\theta > w + 2$.
$(1 - \mathbb{1}_{\lvert\theta\rvert \le w+2}) \cdot (1 - \mathbb{1}_{\theta > 0}) \cdot \mathbb{1}_{\theta_a} \cdot ((s_a - s_c)^2 + \theta_a^2) = 0$;	Otherwise, accept iff $s_a = s_c \wedge e_a = e_c$.
$(1 - \mathbb{1}_{\lvert\theta\rvert \le w+2}) \cdot \mathbb{1}_{\theta > 0} \cdot \mathbb{1}_{\theta_b} = 0$;	Case II is similar to Case I.
$(1 - \mathbb{1}_{\lvert\theta\rvert \le w+2}) \cdot (1 - \mathbb{1}_{\theta > 0}) \cdot \mathbb{1}_{\theta_b} \cdot ((s_b - s_c)^2 + \theta_b^2) = 0$;	
$(1 - \mathbb{1}_{\lvert\theta\rvert \le w+2}) \cdot \mathbb{1}_{\Delta_{a,b}} = 0$.	In case III, always reject.

- Add all above constraints to the R1CS instance $\mathbb{X}$ and update S_1 and S_2 as follows, $S_1 := S_1 \cup \{\{x_j, y_j\}_{j \in \{\mathrm{I}, \mathrm{II}, \mathrm{III}\}}, \{\gamma_k\}_{k \in [3]}\}$, $S_2 := S_2 \cup \{z, \{\Theta_j\}_{j \in \{\mathrm{I}, \mathrm{II}, \mathrm{III}\}}\}$.
- For every multiplication gate g, we introduce the following new variables and constraints:

Constraints for Exponentiations	Notes
$\theta = e_c - (e_a + e_b)$; $(\theta - w)(\theta - w + 1) = 0$.	Reject if $\theta \notin \{w - 1, w\}$.
$\lambda_\theta = (\theta - w + 1) \cdot 2^w - (\theta - w) \cdot 2^{w-1}$.	Otherwise compute 2^θ.

Constraints for Range Proofs	Notes
$x = \delta^{-1} \cdot (s_a \cdot s_b - s_c \cdot \lambda_\theta)$, $y = s_a \cdot s_b$; $z = (x + y)(y - x)$; $4z - 3 = \gamma_1^2 + \gamma_2^2 + \gamma_3^2$.	Apply range proof.

- Add all above constraints to the R1CS instance $\mathbb{X}$ and update S_1 and S_2 as follows, $S_1 := S_1 \cup \{x, y, \{\gamma_k\}_{k \in [3]}\}, S_2 := S_2 \cup \{z\}$.
- Finally, output $(\mathbb{X}, \{(S_1, w_1 := 3w + 6), (S_2, w_2 := 6w + 12)\})$.

6.6 Security Proofs

We defer the security proofs to the full version.

6.7 Optimization and Performance

R1CSCompiler **for Small** p**.** In Section 6.1, we constructed a R1CSCompiler for $\log_2 p > 6w + 3\kappa + 2\log_2 n + O(1)$. In practice, the typical choices of the underlying groups are Curve25519 or BN-256 which leads to $\log_2 p \approx 256$, and BN-384 which leads to $\log_2 p \approx 384$. If one wants to choose 256-bits groups, and use $w = 24$ for the precision of a IEEE 32-bit floating point number, then κ becomes as small as $20 \sim 30$. Hence, we provide a R1CSCompiler that reduces the size of p to $3w + 3\kappa + 2\log_2 n + O(1)$, at the cost of producing a slightly larger R1CS instance.

Our key observation is that, the $6w$ term sources from the value $z_i = (x_i + y_i)(y_i - x_i)$, where x_i, y_i's are both $3w$-bits integers. Therefore, the value z_i has $6w$-bits. Hence, to use a smaller p, we avoid computing $z_i = (x_i + y_i)(y_i - x_i)$. Instead, to prove $(x_i + y_i)(y_i - x_i) > 0$, we firstly compute a helper variable $s_i \in \{-1, +1\}$ as the sign of $x_i + y_i$. Then we prove that $s_i \cdot (y_i - x_i) > 0$. Specifically, we modify the R1CSCompiler in Section 6.1 as follows.

- …
- Compute $s_i \in \{-1, +1\}$ as the sign of $x_i + y_i$. To ensure s_i is computed correctly, we add a constraint $s_i \cdot (x_i + y_i) > 0$ to the R1CS instance $\mathbb{X}$, and convert it to equalities using sum of three squares, and then add a constraint $s_i^2 = 1$ to ensure $s_i \in \{-1, +1\}$.
- Add a constraint $s_i \cdot (y_i - x_i) > 0$ to the R1CS instance $\mathbb{X}$ by sum of three squares.
- …

In terms of our instantiation, we will pick the following parameters for p:

- **"Small groups"** refers to the field size p with $\log_2 p > 3w + 3\kappa + 2\log_2 n + O(1)$. We choose $p \approx 2^{256}$ for small group.
- **"Large groups"** refers to the field size p with $\log_2 p > 6w + 3\kappa + 2\log_2 n + O(1)$. We choose $p \approx 2^{384}$ for large group.

Soundness Amplification. We set the parameter $\kappa \approx 40$ for small $p \approx 2^{256}$. To achieve > 80-bits security, we repeat the random linear combination for 2 times.

Optimized Bit-Decomposition Method. We compare our method with the following method of verifying relative error via bit decomposition: first convert the verification on the upper bound of the relative error to the verification of some value $z \ge 0$ in the same way as before, then we bit-decompose z as $z = \sum_{i=0}^{\lfloor \log z \rfloor} 2^i z_i$, where z_i is the i-th bit of z. Then we use the constraint $z_i \cdot (1 - z_i) = 0$ to encode $z_i \in \{0, 1\}$. In this way, we convert $z \ge 0$ to a R1CS instance.

Performance. We compare our work with other different methods in terms of the following metrics:

- **Size of** R1CS **per constraint**: We count the size of R1CS instance per constraint as the additional *average* number of *non-zero entries* in the matrix A, B and C when, adding such constraint to the R1CS (See the full version for the definition of R1CS. We estimate **Prover Efficiency** based on the total number of *non-zero entries*, times the number of group operations spent on each non-zero entry. In practice, we find that "Large groups" operations are about 2× slower than "Small groups" (See our choice of groups above).

– **Overall proof size**: This is the size of the prover's message size. For any scheme which internally utilizes "commit-and-prove" proof system, we use $|\Pi|$ to denote the proof size, and $|\mathbf{c}|$ to denote the commitment size of the underlying "commit-and-prove" proof system, thus $|\Pi| + |\mathbf{c}|$ being the overall proof size.

We choose the precision $w = 24$ or 53, which corresponds to the precision of an IEEE 32-bit floating point number or 64-bit floating point number, respectively.

Concrete Efficiency. We present concrete efficiency of our *zero-knowledge* commit-and-prove protocol for floating point number addition/multiplication (without sublinear verification). For comparison, we consider two other approaches:

– The "optimized bit decomposition" method used by [9] for range proofs. We denote this method by "Bit." in our tables.
– The method of converting a floating point addition/multiplication to a binary circuit. We denote this method by "IEEE-754" in our tables. For 32-bit floating numbers we list the circuit size in the source code of [35]. For 64-bit floating numbers we list the circuit size achieved by [2].

We first compare the **size of** R1CS **per constraint** of our protocol using "large groups", with other approaches using "small groups". The concrete numbers are presented in table 2.

For 32-bit floating point computation with respect to an arithmetic circuit with even number of addition and multiplication gates, the **size of** R1CS **per gate** of our protocol is 91× less than that of the method following IEEE standard exactly [35] and is 4× less than that of the optimized bit-decomposition method. For 64-bit floating point computation, this size of our protocol is 432× less than that of strictly following IEEE standard and 7× less than that of the optimized bit-decomposition method.

In terms of **prover efficiency**, taking account for the difference in group sizes, for 32-bit floating point computation our protocol is 45× faster than the method following IEEE standard exactly and is 2× faster than the optimized bit-decomposition method. For 64-bit floating point computation, our protocol is 216× faster than IEEE standard and 3.5× faster than the optimized bit-decomposition method.

The overall proof size is approximately 2× that of the other two methods since in our zero-knowledge protocol, we send two commitments $(\mathbf{c}, \mathbf{c}')$ and proofs (Π, Π'). We conservatively estimate the proof Π' having same size as Π. In reality, the instance R1CS′ contains only a small number of non-zero entries, hence Π' should have a smaller size.

		This work	Bit.	IEEE-754
32-bit	$\|$R1CS$\|$ per (+)	89	296	2456
	$\|$R1CS$\|$ per (×)	35	207	8854
	Overall proof size	$2(\|\Pi\| + \|c\|)$	$\|\Pi\| + \|\mathbf{c}\|$	$\|\Pi\| + \|\mathbf{c}\|$
64-bit	$\|$R1CS$\|$ per (+)	115	528	15637
	$\|$R1CS$\|$ per (×)	24	439	44899
	Overall proof size	$2(\|\Pi\| + \|c\|)$	$\|\Pi\| + \|\mathbf{c}\|$	$\|\Pi\| + \|\mathbf{c}\|$

Table 2: Concrete performance for succinct zero-knowledge of floating point number addition/multiplication with linear-time verification in the large groups ($\log_2 p = 384$).

We then compare the performance of our protocol for 32-bit floating point computations, using "small groups", with other approaches using "small groups". The concrete numbers are presented in table 3.

For 32-bit floating point computations, the **size of** R1CS **per gate** of our protocol is 85× less than the method following the IEEE standard exactly [35] and 3.8× less than the optimized bit-decomposition method above. Since all these approaches operate on the same group, these numbers also translate to **prover efficiency**.

		This work	Bit.	IEEE-754
32-bit	$\|$R1CS$\|$ per (+)	108	296	2456
	$\|$R1CS$\|$ per (×)	25	207	8854
	Overall proof size	$2(\|\Pi\| + \|\mathbf{c}\|)$	$\|\Pi\| + \|\mathbf{c}\|$	$\|\Pi\| + \|\mathbf{c}\|$

Table 3: Concrete performance for succinct zero-knowledge of floating point number addition/multiplication with linear-time verification in the small groups ($\log_2 p = 256$).

Additionally we compare our succinct argument of floating point number computation with sub-linear verification (without zero-knowledge) with the optimized bit-decomposition method. In this case the prover efficiency of our work only increases 53% for 32-bit computation and 28% for 64-bit computation.

Instantiations We defer instantiation of our protocol to the full version.

7 CONCLUSION

In this work, we study the design of succinct ZK proof systems for floating point computations. We provide two contributions: a new relative error model for verifying floating point computations, and an efficient succinct ZK proof system with sublinear verification.

Our work motivates several interesting directions for future work. First, reducing the verification complexity to poly-logarithmic would be very useful. Another interesting direction is modeling and constructing efficient succinct proof systems for applications where the correct rounding is crucial, such as finance and accounting.

ACKNOWLEDGMENTS

The first and fourth authors were supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by the Sloan Foundation, and Visa Inc. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

The second and third authors were supported in part by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. The second author was additionally supported in part by AFOSR Award FA9550-19-1-0200 and the Office of Naval Research Grant N00014-19-1-2294. This work was done while the second and third authors were visiting University of California Berkeley.

REFERENCES

[1] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 2087–2104. https://doi.org/10.1145/3133956.3134104

[2] David W. Archer, Shahla Atapoor, and Nigel P. Smart. 2021. The Cost of IEEE Arithmetic in Secure Computation. LatinCrypt. https://ia.cr/2021/054.

[3] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM CCS 93*, Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby (Eds.). ACM Press, 62–73. https://doi.org/10.1145/168588.168596

[4] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 459–474. https://doi.org/10.1109/SP.2014.36

[5] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. 2019. Aurora: Transparent Succinct Arguments for R1CS. In *EUROCRYPT 2019, Part I (LNCS, Vol. 11476)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, Heidelberg, 103–128. https://doi.org/10.1007/978-3-030-17653-2_4

[6] Flávio Bergamaschi, Shai Halevi, Tzipora T. Halevi, and Hamish Hunt. 2019. Homomorphic Training of 30,000 Logistic Regression Models. In *ACNS 19 (LNCS, Vol. 11464)*, Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung (Eds.). Springer, Heidelberg, 592–611. https://doi.org/10.1007/978-3-030-21568-2_29

[7] Avrim Blum, John Hopcroft, and Ravindran Kannan. 2020. *Foundations of Data Science*. Cambridge University Press. https://doi.org/10.1017/9781108755528

[8] Fabrice Boudot. 2000. Efficient Proofs that a Committed Number Lies in an Interval. In *EUROCRYPT 2000 (LNCS, Vol. 1807)*, Bart Preneel (Ed.). Springer, Heidelberg, 431–444. https://doi.org/10.1007/3-540-45539-6_31

[9] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 315–334. https://doi.org/10.1109/SP.2018.00020

[10] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. 2020. Transparent SNARKs from DARK Compilers. In *EUROCRYPT 2020, Part I (LNCS, Vol. 12105)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Heidelberg, 677–706. https://doi.org/10.1007/978-3-030-45721-1_24

[11] Jan Camenisch, Rafik Chaabouni, and abhi shelat. 2008. Efficient Protocols for Set Membership and Range Proofs. In *ASIACRYPT 2008 (LNCS, Vol. 5350)*, Josef Pieprzyk (Ed.). Springer, Heidelberg, 234–252. https://doi.org/10.1007/978-3-540-89255-7_15

[12] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *ASIACRYPT 2017, Part I (LNCS, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, Heidelberg, 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

[13] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. 2020. Fractal: Post-quantum and Transparent Recursive Proofs from Holography. In *EUROCRYPT 2020, Part I (LNCS, Vol. 12105)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Heidelberg, 769–793. https://doi.org/10.1007/978-3-030-45721-1_27

[14] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. 2012. Practical verified computation with streaming interactive proofs. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, 90–112. https://doi.org/10.1145/2090236.2090245

[15] Geoffroy Couteau, Michael Klooß, Huang Lin, and Michael Reichle. 2021. Efficient Range Proofs with Transparent Setup from Bounded Integer Commitments. In *EUROCRYPT 2021, Part III (LNCS, Vol. 12698)*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, Heidelberg, 247–277. https://doi.org/10.1007/978-3-030-77883-5_9

[16] Geoffroy Couteau, Thomas Peters, and David Pointcheval. 2017. Removing the Strong RSA Assumption from Arguments over the Integers. In *EUROCRYPT 2017, Part II (LNCS, Vol. 10211)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.). Springer, Heidelberg, 321–350. https://doi.org/10.1007/978-3-319-56614-6_11

[17] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86 (LNCS, Vol. 263)*, Andrew M. Odlyzko (Ed.). Springer, Heidelberg, 186–194. https://doi.org/10.1007/3-540-47721-7_12

[18] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*, Michael Mitzenmacher (Ed.). ACM Press, 169–178. https://doi.org/10.1145/1536414.1536440

[19] David Goldberg. 1991. What Every Computer Scientist Should Know about Floating-Point Arithmetic. *ACM Comput. Surv.* 23, 1 (mar 1991), 5–48. https://doi.org/10.1145/103162.103163

[20] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. 2015. Delegating Computation: Interactive Proofs for Muggles. *J. ACM* 62, 4, Article 27 (sep 2015), 64 pages. https://doi.org/10.1145/2699436

[21] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *17th ACM STOC*. ACM Press, 291–304. https://doi.org/10.1145/22145.22178

[22] Jens Groth. 2005. Non-interactive Zero-Knowledge Arguments for Voting. In *ACNS 05 (LNCS, Vol. 3531)*, John Ioannidis, Angelos Keromytis, and Moti Yung (Eds.). Springer, Heidelberg, 467–482. https://doi.org/10.1007/11496137_32

[23] Jens Groth. 2011. Efficient Zero-Knowledge Arguments from Two-Tiered Homomorphic Commitments. In *ASIACRYPT 2011 (LNCS, Vol. 7073)*, Dong Hoon Lee and Xiaoyun Wang (Eds.). Springer, Heidelberg, 431–448. https://doi.org/10.1007/978-3-642-25385-0_23

[24] Joe Kilian. 1992. A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract). In *24th ACM STOC*. ACM Press, 723–732. https://doi.org/10.1145/129712.129782

[25] Helger Lipmaa. 2003. On Diophantine Complexity and Statistical Zero-Knowledge Arguments. In *ASIACRYPT 2003 (LNCS, Vol. 2894)*, Chi-Sung Laih (Ed.). Springer, Heidelberg, 398–415. https://doi.org/10.1007/978-3-540-40061-5_26

[26] Silvio Micali. 2000. Computationally Sound Proofs. *SIAM J. Comput.* 30, 4 (2000), 1253–1298. https://doi.org/10.1137/S0097539795284959 arXiv:https://doi.org/10.1137/S0097539795284959

[27] J. O. Rabin and Jeffrey Shallit. 1985. *Randomized Algorithms in Number Theory*. Technical Report. USA.

[28] Michael O. Rabin and Jeffery O. Shallit. 1986. Randomized algorithms in number theory. *Communications on Pure and Applied Mathematics* 39, S1 (1986), S239–S256. https://doi.org/10.1002/cpa.3160390713 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpa.3160390713

[29] Srinath Setty. 2020. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *CRYPTO 2020, Part III (LNCS, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, Heidelberg, 704–737. https://doi.org/10.1007/978-3-030-56877-1_25

[30] Srinath Setty and Jonathan Lee. 2020. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275. https://ia.cr/2020/1275.

[31] Srinath T. V. Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. 2012. Taking Proof-Based Verified Computation a Few Steps Closer to Practicality. In *USENIX Security 2012*, Tadayoshi Kohno (Ed.). USENIX Association, 253–268.

[32] Justin Thaler. 2013. Time-Optimal Interactive Proofs for Circuit Evaluation. In *CRYPTO 2013, Part II (LNCS, Vol. 8043)*, Ran Canetti and Juan A. Garay (Eds.). Springer, Heidelberg, 71–89. https://doi.org/10.1007/978-3-642-40084-1_5

[33] Lloyd N. Trefethen. 2008. *IV. 21 Numerical Analysis, in The Princeton Companion to Mathematics* (illustrated edition ed.). Princeton University Press, USA.

[34] Lloyd N. Trefethen and David Bau. 1997. *Numerical Linear Algebra*. SIAM.

[35] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. Cryptology ePrint Archive, Report 2021/730. https://ia.cr/2021/730.

[36] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation. In *CRYPTO 2019, Part III (LNCS, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, Heidelberg, 733–764. https://doi.org/10.1007/978-3-030-26954-8_24

[37] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. 2021. Doubly Efficient Interactive Proofs for General Arithmetic Circuits with Linear Prover Time. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 159–177. https://doi.org/10.1145/3460120.3484767

[38] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2020. Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof. In *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 859–876. https://doi.org/10.1109/SP40000.2020.00052