Divisions and Square Roots with Tight Error Analysis from Newton–Raphson Iteration in Secure Fixed-Point Arithmetic

cryptography

MDPI

Article

Divisions and Square Roots with Tight Error Analysis from Newton–Raphson Iteration in Secure Fixed-Point Arithmetic

Stan Korzilius and Berry Schoenmakers *

Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

* Correspondence: l.a.m.schoenmakers@tue.nl

Abstract: In this paper, we present new variants of Newton–Raphson-based protocols for the secure computation of the reciprocal and the (reciprocal) square root. The protocols rely on secure fixed-point arithmetic with arbitrary precision parameterized by the total bit length of the fixed-point numbers and the bit length of the fractional part. We perform a rigorous error analysis aiming for tight accuracy claims while minimizing the overall cost of the protocols. Due to the nature of secure fixed-point arithmetic, we perform the analysis in terms of absolute errors. Whenever possible, we allow for stochastic (or probabilistic) rounding as an efficient alternative to deterministic rounding. We also present a new protocol for secure integer division based on our protocol for secure fixed-point reciprocals. The resulting protocol is parameterized by the bit length of the inputs and yields exact results for the integral quotient and remainder. The protocol is very efficient, minimizing the number of secure comparisons. Similarly, we present a new protocol for integer square roots based on our protocol for secure fixed-point square roots. The quadratic convergence of the Newton–Raphson method implies a logarithmic number of iterations as a function of the required precision (independent of the input value). The standard error analysis of the Newton–Raphson method focuses on the termination condition for attaining the required precision, assuming sufficiently precise floating-point arithmetic. We perform an intricate error analysis assuming fixed-point arithmetic of minimal precision throughout and minimizing the number of iterations in the worst case.

Keywords: multiparty computation; error analysis; fixed-point arithmetic; reciprocal; integer division; reciprocal square root; (integer) square root

Citation: Korzilius, S.; Schoenmakers, B. Divisions and Square Roots with Tight Error Analysis from Newton–Raphson Iteration in Secure Fixed-Point Arithmetic. *Cryptography* **2023**, *7*, 43. https://doi.org/10.3390/cryptography7030043

Academic Editor: Shay Gueron

Received: 11 July 2023
Revised: 7 September 2023
Accepted: 8 September 2023
Published: 12 September 2023

1. Introduction

In this paper, we design and analyze protocols for secure fixed-point arithmetic as a practical alternative to secure floating-point arithmetic. From a numerical analysis perspective, floating-point arithmetic is very useful as floating-point numbers scale dynamically and relative errors can be controlled appropriately. Performance-wise, however, secure floating-point arithmetic is very demanding. Compared to secure integer arithmetic, for example, full support for secure floating-point numbers is usually orders of magnitude more costly. This holds across many frameworks for secure computation, ranging from all flavors of multiparty computation to fully homomorphic encryption and indistinguishability obfuscation.

Secure fixed-point arithmetic strikes a balance between performance and usability. Addition/subtraction is as efficient as for integers, whereas multiplication is costlier but relatively straightforward. Our focus in this paper is on more advanced operations such as division (via the reciprocal) and taking square roots. We present new protocols based on Newton–Raphson iteration, along with a detailed error analysis for strict accuracy guarantees at minimal cost. Moreover, turning the tables around, we show how to obtain efficient protocols for secure integer division and integer square roots from their secure fixed-point counterparts.

The Newton–Raphson method has been studied extensively in the literature on secure fixed-point arithmetic, starting with the paper by Algesheimer et al. [1]. This important paper contained the groundwork for the secure computation of the reciprocal, including a thorough error analysis, in fact aimed at a direct application to secure integer division. The works by Catrina et al. [2,3] presented the basic foundation for secure fixed-point arithmetic, also introducing probabilistic rounding as an efficient alternative to deterministic rounding. In this paper, we will closely follow the Newton–Raphson-based protocol for the reciprocal from [2]. However, we will fine-tune the use of probabilistic vs. deterministic rounding, limiting the number of truncated bits as much as possible, to guarantee an absolute error below 2^{-f} for any desired precision of f fractional bits.

In this paper, we will also use the Newton–Raphson method for the secure computation of the reciprocal square root. Prior work by Liedel [4] and follow-up work by Aly and Smart [5] used Goldschmidt's method for the reciprocal square root. However, these papers lacked a complete error analysis and did not guarantee an absolute error below 2^{-f} for any desired precision of f fractional bits. In this paper, we will present a fine-tuned secure protocol for the reciprocal square root and a detailed error analysis, following the same approach as for the reciprocal. We will also extend this protocol to compute the square root, with the same guarantee for the absolute error.

We note that the error analysis of applications of the Newton–Raphson method commonly focuses on bounds for the relative error assuming floating-point arithmetic. Research into the accuracy of fixed-point arithmetic is in general rather limited. Sources like [6–8] (Section 4.2, in particular) have treated some basic aspects. For instance, Wilkinson [7] already covered the basic idea that the inner product of two vectors $\mathbf{x}$, $\mathbf{y}$ can be computed accurately by accumulating the terms $x_i y_i$ exactly and only rounding the final sum to the desired precision; this idea carries over to the setting of secure computation, see Table 2 in [2]. A further aspect of the secure use of the Newton–Raphson method is that it should always be run for the same (worst-case) number of iterations to avoid leaking information about the input. In this paper, we present the first detailed analysis taking all these aspects into account.

We present our solutions in a generic way, assuming secure integer and fixed-point arithmetic with a small set of basic operations. Each basic operation needs to be implemented by means of a secure protocol, operating on either secret-shared, encrypted, or encoded values, depending on the underlying framework for secure computation. Although the performance of these protocols varies across frameworks, the relative performance behaves similarly between operations like secure addition, multiplication, or comparison, as well as the secure generation of random bits. For concreteness, we will focus on secure multiparty computation (MPC) as the underlying framework. Specifically, we consider the use of probabilistic rounding (versus deterministic rounding) to limit the cost of secure fixed-point multiplications. Many results, however, carry over to related areas in cryptography such as (fully) homomorphic encryption.

The paper is organized as follows. Above, we elaborated on the state of the art for the secure fixed-point computation of the reciprocal and the reciprocal square root, emphasizing the lack of detailed error analyses. Section 2 explains some basic aspects of MPC and provides a brief introduction to secure fixed-point arithmetic; in particular, some details about the use of probabilistic rounding are presented, and the basics of the Newton–Raphson method are highlighted. Section 3 presents our solution for the secure computation of the reciprocal, together with a tight error analysis achieving a given fixed-point precision while minimizing the computational cost. In Section 4, we demonstrate a direct application of the secure fixed-point reciprocal, namely for efficient secure integer division (with the remainder). Section 5 then presents our solution for the secure computation of the reciprocal square root, essentially following the same approach as for the reciprocal, although the details are more intricate. In Section 6 we demonstrate a direct application of the secure fixed-point reciprocal square root, namely for the efficient secure computation of fixed-point square roots with precise error bounds, which we use in

turn for efficient secure integer square roots. We conclude in Section 7 and mention some applications and concurrent work on a related problem. Finally, Appendix A collects all lemmas and proofs left out of the main text; all theorems and proofs are included in the main text.

2. Preliminaries

Below, we provide the background on secure fixed-point arithmetic underlying all protocols in this paper. We also discuss the concept of probabilistic rounding and briefly review the Newton–Raphson method.

2.1. Secure Computation

We present our protocols for the secure computation of the reciprocal and the (reciprocal) square root in terms of an arithmetic black box (following, e.g., [9–11]). The protocols are specified in pseudocode, using a limited set of operations commonly supported in many MPC frameworks. The parties executing these operations are suppressed from the notation.

We use $[\![a]\!]$ to denote a secure representation of value a. That is, $[\![a]\!]$ can be thought of as either a secret-shared value a or a public-key (homomorphic) encryption of a value a. We let $\mathsf{Open}([\![a]\!])$ denote the pooling of (decryption) shares to reveal the value of a in the clear. Secure arithmetic over a finite field (or finite ring) using $+, -, *, /$ is assumed to be available. The common representation of integral and fixed-point numbers as ℓ-bit integers in a bounded range $[-2^{\ell-1}, 2^{\ell-1}) \subset \mathbb{Z}_N$ is assumed, where $2^{\ell+\kappa} < N$ for security parameter κ. This allows for efficient secure comparisons $<, \leq, >, \geq, =, \neq$. To denote a uniform randomly generated secure bit b, we write $[\![b]\!] \in_R \{0,1\}$. Similarly, we write $[\![r]\!] \in_R \{0,1,\ldots,2^{\ell+\kappa}-1\}$ to denote a secure integer r distributed sufficiently randomly such that the statistical distance $\Delta(r; 2^\ell + r) < 2^{-\kappa}$ is negligible as a function of κ.

As a more advanced primitive, we assume the availability of operation $[\![v]\!] \leftarrow \mathsf{Scale}([\![a]\!])$, for $a \neq 0$. Here, $v = \pm 2^k$, for some $k \in \mathbb{Z}$, is uniquely determined by the constraint $\frac{1}{2} \leq av < 1$. Similarly, we use $[\![v]\!], [\![v^{\frac{1}{2}}]\!] \leftarrow \mathsf{Scale}([\![a]\!])$ to denote the same operation with the additional constraint that k is even.

Efficient implementations for these operations are assumed. The round complexity is typically either constant or logarithmic. To ensure logarithmic round complexity of $O(\log \ell)$ for our protocols operating on ℓ-bit fixed-point numbers, it suffices that basic secure arithmetic $+, -, *, /$ takes $O(1)$ rounds and that secure comparison $<$ and $\mathsf{Scale}([\![a]\!])$ take $O(\log \ell)$ rounds.

2.2. Secure Fixed-Point Arithmetic

We follow the model for secure fixed-point arithmetic put forth by Catrina et al. [2,3]. For $\ell > f \geq 0$, the set $\mathbb{Q}_{\ell,f}$ of ℓ-bit fixed-point numbers with f fractional bits is defined as

$$\mathbb{Q}_{\ell,f} = \{\overline{x}\, 2^{-f} : \overline{x} \in \mathbb{Z}, -2^{\ell-1} \leq \overline{x} < 2^{\ell-1}\}.$$

The integer part of a fixed-point number thus consists of $e = \ell - f$ bits, of which the most significant bit represents the sign. Phrased differently, we use two's complement for the binary representation of fixed-point numbers $x \in \mathbb{Q}_{\ell,f}$:

$$x = (d_{e-1} \ldots d_0 . d_{-1} \ldots d_{-f})_2 = -d_{e-1} 2^{e-1} + \sum_{i=-f}^{e-2} d_i 2^i, \qquad \text{with } d_i \in \{0,1\}.$$

The value $\delta_f = 2^{-f}$ corresponding to the least significant bit of x is also loosely referred to as the precision.

For the implementation of fixed-point arithmetic, a number $x = \overline{x}\, 2^{-f} \in \mathbb{Q}_{\ell,f}$ is simply represented by the integer $\overline{x}$. This integer representation is particularly convenient for the implementation of secure fixed-point arithmetic, e.g., when all computation is carried out with secret-shared numbers over a prime field. The factor 2^{-f} is publicly known and is

only used when the results are output as fixed-point numbers. The actual calculations are performed with integer values only.

The sum of two fixed-point numbers x and y is obtained by adding their integer representations. That is, setting $\overline{x+y} = \overline{x} + \overline{y}$ gives the correct result:

$$\overline{x+y}\,2^{-f} = (\overline{x}+\overline{y})\,2^{-f} = \overline{x}\,2^{-f} + \overline{y}\,2^{-f} = x + y.$$

For the product of a fixed-point number x and an integer t, we set $\overline{tx} = t\,\overline{x}$ to obtain the desired result:

$$\overline{tx}\,2^{-f} = (t\,\overline{x})\,2^{-f} = t(\overline{x}\,2^{-f}) = tx.$$

Computing the product of two fixed-point numbers, however, is slightly more involved. Simply multiplying the integer representations $\overline{x}$ and $\overline{y}$ does not yield a useful result for $\overline{xy}$:

$$\left|\overline{x}\,\overline{y}\,2^{-f} - xy\right| = \left|(x2^f)(y2^f)\,2^{-f} - xy\right| = \left|xy2^f - xy\right| \gg 0.$$

We therefore divide $\overline{x}\,\overline{y}$ by 2^f and apply some form of rounding to obtain an integral result. For instance, we may use $\lfloor \overline{x}\,\overline{y}\,2^{-f} \rceil$ as a close approximation of $\overline{xy}$, where $\lfloor\cdot\rceil$ denotes rounding to the nearest integer:

$$\left|\lfloor \overline{x}\,\overline{y}\,2^{-f} \rceil\,2^{-f} - xy\right| = \left|\lfloor xy\,2^f \rceil\,2^{-f} - xy\right| = \left|\lfloor xy\,2^f \rceil - xy\,2^f\right|2^{-f} \leq \tfrac{1}{2}2^{-f} = \tfrac{1}{2}\delta_f.$$

By (deterministically) rounding to the nearest integer, the absolute error is limited to $\frac{1}{2}\delta_f$ in the worst case. For reasons of efficiency, however, we will often allow a slightly larger error of δ_f in the worst case by using probabilistic rounding.

Remark 1. *In the remainder of this paper, we will use the integer representation of fixed-point numbers in the pseudocode of the algorithms. For a better intuitive understanding, however, we consider the actual fixed-point numbers in the error analyses. Concretely, if we write x, this means x in the analyses but $\overline{x}$ in the algorithms.*

2.3. Probabilistic Rounding

Apart from the primitives for secure computation introduced above, we will use two specific methods for rounding secure fixed-point numbers. Algorithm 1 covers both methods, referred to as deterministic and probabilistic rounding, respectively. Deterministic rounding is the common method of rounding a to the nearest integer $\lfloor a \rceil$. Probabilistic rounding [2,3] yields either $\lfloor a \rfloor$ or $\lceil a \rceil$ as a result, where the value closest to a tends to be more likely.

Remark 2. *Probabilistic (or stochastic) rounding is applied in various research fields, including machine learning, ODEs and PDEs, quantum computing, and digital signal processing, usually in combination with a severe limitation on numerical precision (see, for instance, [12–15]). The latter condition makes probabilistic rounding desirable in these cases, because it ensures zero-mean rounding errors and avoids the problem of stagnation, where small values are lost to rounding when they are added to an increasingly large accumulator [16]. However, the use of a randomness source may be expensive, as the number of random bits (entropy) varies with the probability distribution required for the rounding errors.*

To make the distinction between deterministic rounding and probabilistic rounding more concrete, consider the following equation for the exact result of the product xy:

$$xy\,2^f = \lfloor xy\,2^f \rfloor + r.$$

The first term on the right-hand side captures the integer part of xy together with the first f fractional bits, while r contains the remaining f fractional bits; hence, $r \in [0,\ 1)$. The probabilistically rounded result $\lfloor xy \rceil_{\$}$ then yields

$$\lfloor xy \rceil_{\$} = \begin{cases} xy - r\,\delta_f & \text{with probability } 1 - r, \\ xy + (1 - r)\,\delta_f & \text{with probability } r. \end{cases}$$

The maximum difference δ_f between xy and $\lfloor xy \rceil_{\$}$ occurs when $xy = \lfloor xy \rfloor$ and $\lfloor xy \rceil_{\$} = \lfloor xy \rfloor + \delta_f$, hence only when $r = 0$. This happens with probability 0, so for the *probabilistic rounding* error e after a single multiplication, we have $|e| < \delta_f$. As always, for the *deterministic rounding* error e, we have $|e| \leq \frac{1}{2}\delta_f$.

As can be seen from Algorithm 1, deterministic rounding in MPC is significantly more expensive than probabilistic rounding due to the use of the secure comparison $c' < [\![r]\!]$ in line 10. Given the bits $[\![r_0]\!], \ldots, [\![r_{\nu-1}]\!]$ of $[\![r]\!]$ and the bits of c', a common implementation of $c' < [\![r]\!]$ takes approximately ν secure multiplications in $\log_2 \nu$ rounds, whereas the other parts of the algorithm commonly take $O(1)$ rounds (the asymptotic round complexity for secure comparison can be limited to $O(1)$ rounds following [10], but the hidden constant is too large for practical purposes). For the deterministic rounding of $a/2^\nu$ to the nearest integer, we first add $2^{\nu-1}$ to a and then truncate the ν least significant bits. The comparison $c' < [\![r]\!]$ is needed to obtain the correct output. For probabilistic rounding, we omit the corrections in lines 2 and 10, saving the work for a secure comparison.

Algorithm 1 $\mathsf{Round}_\nu([\![a]\!], \text{mode} = \text{probabilistic})$ $\quad -2^{\ell+\nu-1} \leq a < 2^{\ell+\nu-1}$

1: **if** mode = deterministic **then**
2: $\quad [\![a]\!] \leftarrow [\![a]\!] + 2^{\nu-1}$
3: $[\![r_0]\!], \ldots, [\![r_{\nu-1}]\!] \in_R \{0,1\}$ ▷ ν random bits
4: $[\![r]\!] \leftarrow \sum_{i=0}^{\nu-1} [\![r_i]\!] 2^i$
5: $[\![r']\!] \in_R \{0,1,\ldots,2^{\kappa+\ell}-1\}$ ▷ security parameter κ
6: $c \leftarrow \mathsf{Open}(2^{\ell-1+\nu} + [\![a]\!] + [\![r]\!] + 2^\nu [\![r']\!])$
7: $c' \leftarrow c \mod 2^\nu$
8: $[\![b]\!] \leftarrow ([\![a]\!] + [\![r]\!] - c') \,/\, 2^\nu$ ▷ $b = \lfloor a/2^\nu \rceil_{\$}$
9: **if** mode = deterministic **then**
10: $\quad [\![b]\!] \leftarrow [\![b]\!] - (c' < [\![r]\!])$ ▷ $b = \lfloor a/2^\nu \rceil$
11: **return** $[\![b]\!]$ ▷ $-2^{\ell-1} \leq b < 2^{\ell-1}$

2.4. Newton–Raphson Method

The Newton–Raphson method (also known as Newton's method) is a numerical procedure to find roots of functions. The method has been known for centuries and extensively studied and analyzed in the literature (see, e.g., ref. [17] for a general description of the method and [18] for a historical overview of its convergence properties). Without providing further details on the derivation, we simply state that, given an approximation c_i to the root of a function $f \equiv f(c)$, better approximations may be found in an iterative fashion using the update formula:

$$c_{i+1} = c_i - \frac{f(c_i)}{f'(c_i)}. \tag{1}$$

There are a few conditions that must be satisfied for the Newton–Raphson method to work. For the moment, it suffices to say that an important aspect of the method is that it requires an initial approximation, which needs to be of sufficient accuracy.

3. Reciprocal

In this section, we consider the secure computation of the reciprocal using the Newton–Raphson method. This approximation of the reciprocal will also serve as a basis for secure integer division in Section 4.

We perform a tight error analysis to guarantee an absolute error not exceeding $\delta_f = 2^{-f}$ while minimizing the additional precision used during the computation.

3.1. Secure Computation

The reciprocal function evaluates $[\![1/a]\!]$ for a secret-shared value $[\![a]\!]$, $a \neq 0$, see Algorithm 2. As a first step towards a good initial approximation, $[\![a]\!]$ is scaled to $[\![b]\!] = [\![a]\!][\![v]\!]$, where $v = \pm 2^k$ for some $k \in \mathbb{Z}$. The scaling factor v is chosen such that $b \in [0.5,\ 1)$. In this interval, the line $3 - 2b$ is a good approximation for $1/b$, with equality at the endpoints $b = 0.5$ and $b = 1$, and the maximum error occurring at $b = 1/\sqrt{2}$. Shifting this line a distance of half the maximum error downward halves the maximum (absolute) error and results in the initial approximation (as in [3], which in turn relies on [19]):

$$[\![c_0]\!] = 3 - \alpha - 2[\![b]\!], \tag{2}$$

with $\alpha = 3/2 - \sqrt{2} \approx 0.085786$. Compared to $1/b$, c_0 has a maximum error of α (at $b = 0.5$, $b = 1/\sqrt{2}$, $b = 1$). The constant term $3 - \alpha$ may be truncated to whatever precision is used in the computations. The multiplication of $[\![b]\!]$ by 2 is essentially free, as it can be performed locally by the parties without truncation.

Algorithm 2 Reciprocal$([\![a]\!], n = 0)$ $\quad -2^{\ell-1} \leq a < 2^{\ell-1}$

1: $[\![v]\!] \leftarrow \mathsf{Scale}([\![a]\!])$ ▷ $v = \pm 2^k, k \in \mathbb{Z}$
2: $[\![b]\!] \leftarrow \mathsf{Round}_{f-n}([\![a]\!][\![v]\!])$ ▷ $2^{f+n-1} \leq b < 2^{f+n}$
3: $\alpha \leftarrow 3/2 - \sqrt{2}$
4: $\theta \leftarrow \lceil \log_2 \log_\alpha 2^{-(f+n)} \rceil$
5: $[\![c_0]\!] \leftarrow (3 - \alpha)2^f - 2[\![b]\!]$
6: **for** $i = 1$ **to** θ **do**
7: $\quad [\![z]\!] \leftarrow 2 - \mathsf{Round}_{f+n}([\![c_{i-1}]\!][\![b]\!])$
8: $\quad [\![c_i]\!] \leftarrow \mathsf{Round}_{f+n}([\![c_{i-1}]\!][\![z]\!])$
9: $[\![d_\theta]\!] \leftarrow \mathsf{Round}_{f+n}([\![c_\theta]\!][\![v]\!], \text{deterministic})$
10: **return** $[\![d_\theta]\!]$ ▷ $-2^{\ell-1} \leq d_\theta < 2^{\ell-1}$

Given the initial approximation, successive approximations are then computed using

$$[\![c_{i+1}]\!] = [\![c_i]\!](2 - [\![c_i]\!][\![b]\!]), \tag{3}$$

which is obtained by instantiating the Newton–Raphson method in (1) with $f(c) = b - 1/c$.

After θ iterations, with θ independent of the input value a, the final approximation for $1/a$ is obtained from $c_\theta \approx 1/(av)$ as follows:

$$[\![d_\theta]\!] = [\![c_\theta]\!][\![v]\!].$$

The required number of iterations θ will be determined below such that the final error for c_θ does not exceed $\delta_f = 2^{-f}$, assuming exact arithmetic. Subsequently, we will determine the required number of additional bits n for Algorithm 2, taking into account all (rounding) errors. For better readability, we will drop the secret-shared brackets in the remainder of this section.

Under the right circumstances, the Newton–Raphson method converges quadratically to the (nearest) root of a given function, as we will show next. First, with $c = 1/b$, define $\epsilon_i = c - c_i$ as the iteration error. Then, applying (3) and assuming exact arithmetic, we find

$$\epsilon_{i+1} = c - c_i(2 - c_i b) = c - (c - \epsilon_i)(2 - (c - \epsilon_i)b) = \epsilon_i^2 b. \tag{4}$$

Since $b \in [0.5, 1)$ and $|\epsilon_0| \leq \alpha < 1$, we see that quadratic convergence is guaranteed from the start:

$$|\epsilon_i| = b^{2^i - 1}|\epsilon_0|^{2^i} \leq \alpha^{2^i}. \tag{5}$$

To achieve $|\epsilon_\theta| \leq \delta_f$, we thus set

$$\theta = \left\lceil \log_2 \log_\alpha \delta_f \right\rceil. \tag{6}$$

Remark 3. *The behavior at $b = 1$ determines the number of iterations. This observation motivates changing the slope and offset of the linear initial approximation such that the error is slightly smaller at $b = 1$ than it is at $b = 0.5$. If the difference is not too large, the solution at $b = 0.5$ will "catch up" with the solution at $b = 1$ within a certain number of iterations. In some cases, this may save an iteration. For instance, the initial approximation*

$$[\![\varsigma_0]\!] = 2.8312530517578125 - 1.890625[\![b]\!]$$

saves an iteration for various values of $f \geq 29$, including the most common choices for f in this range, namely $f = 2^n$ with $n \in [5, 10]$. For these larger values of f, rounding is most expensive, and thus saving iterations is most valuable. The approximation ς_0 comes with two disadvantages. Firstly, b is multiplied by a number with six fractional digits, instead of an integer; still, the approximation is more efficient in those cases where an iteration is saved. Secondly, the required number of iterations is not as straightforward to compute as it is for c_0, because it is no longer determined by the situation at a single point. With ς_0, the largest error is generally attained at a point close to the middle of $[0.5, 1)$, which slowly shifts to the right for larger values of f.

We further note that quadratic polynomials are also an option. For instance, the following approximation is quite accurate and behaves well during the Newton–Raphson process:

$$[\![\omega_0]\!] = 3[\![b]\!]^2 - 6.5[\![b]\!] + 4.51425.$$

Quadratic polynomials are more expensive due to the computation of $[\![b]\!]^2$, which cannot be performed locally. This makes using quadratic polynomials only worthwhile when it saves iterations—and thus multiplications—in the computations that follow. Unfortunately, this is only true for relatively low values of f, in which cases we save exactly one multiplication in the entire computation. For higher values of f, despite leading to more accurate intermediate approximations, the same number of iterations is required, and hence there is no gain. Because of this, we will not study quadratic polynomials any further and stick to the simpler linear functions. Moreover, to keep things simple in our algorithms and analyses, we will stick to the approximation in (2).

3.2. Tight Error Analysis without Scaling

In this section, we analyze the error $\epsilon_\theta = c - c_\theta$ in the computation of $c = 1/b$ for $b \in [0.5, 1)$. We determine a tight bound for $|\epsilon_\theta|$ taking into account all (rounding) errors, assuming fixed-point arithmetic with f fractional bits in Algorithm 2 (i.e., with $n = 0$). In Section 3.3, we will use this bound to determine the minimal number of additional bits n needed to guarantee that the absolute error for $1/a$ is limited to $\delta_f = 2^{-f}$, also taking into account the errors due to scaling.

Because we use probabilistic rounding in Algorithm 2, each iteration adds a rounding error of $3\delta_f$ in the worst case, see Lemma A2. Due to the quadratic convergence, however, the influence of these rounding errors is limited for subsequent iterations. With the help of Lemma A1, which bounds the error $|\epsilon_{\theta-1}|$ for the penultimate iteration, we are able to give a tight bound for the total error after θ iterations.

Theorem 1. *If the Newton–Raphson method is used to compute* $1/b$ *for* $b \in [0.5, 1)$*, employing initial approximation* (2)*, and computing the number of iterations* θ *with* (6)*, then* $|\epsilon_\theta| < \rho\delta_f$*, where* $\rho = 3.05$*.*

Proof. Clearly, if $\theta = 0$, the initial error is already below δ_f. Because no iterations are performed, no further errors are introduced, and the final error remains below δ_f.

For the cases in which $\theta = 1$ or $\theta = 2$, we exhaustively compute the error for all possible inputs b, considering all rounding possibilities. This covers the values $4 \le f \le 14$ and yields a maximum value for $|\epsilon_\theta|$ of approximately $2.88\delta_f$.

For larger values of f, we follow a different approach: firstly, we derive an expression that bounds the absolute error as a function of f and θ. Secondly, we compute the value of the error bound for $f = 15$ ($\theta = 3$), which will be below $3.05\delta_f$. Thirdly, we show that for larger values of f, the value of the error bound will always be smaller than in the case $f = 15$.

From Lemma A1, we know that in the case of exact arithmetic, the error at the start of the final iteration is bounded by $b^{2^{\theta-1}-1}\sqrt{\delta_f}$. Lemma A2 tells us that in the first iteration, the rounding error is bounded by $(c_0 + 1)\delta_f$, while in every subsequent iteration it is bounded by $(1/b + 1)\delta_f$. Thus, for $\theta \ge 3$, we obtain the following bound for the total error at the start of the final iteration:

$$|\epsilon_{\theta-1}| < b^{2^{\theta-1}-1}\sqrt{\delta_f} + (c_0 + 1)\delta_f + (\theta - 2)\left(\frac{1}{b} + 1\right)\delta_f.$$

Let $T_\theta = T_\theta(b) = (c_0 + 1) + (\theta - 2)(1/b + 1)$. Applying (A2) with $i = \theta - 1$ gives

$$|\epsilon_\theta| < \epsilon_{\theta-1}^2 b + \left(\frac{1}{b} + 1\right)\delta_f. \tag{7}$$

Hence, as an upper bound for $|\epsilon_\theta|$ we get

$$E_{\theta,f}(b)\,\delta_f \stackrel{\text{def}}{=} \left(b^{2^\theta-1} + 2b^{2^{\theta-1}}T_\theta\sqrt{\delta_f} + bT_\theta^2\delta_f + \frac{1}{b} + 1\right)\delta_f.$$

For the case $f = 15$, where $\theta = 3$, this yields

$$E_{3,15}(b)\,\delta_f = \left(b^7 + 2b^4T_3\sqrt{\delta_{15}} + bT_3^2\delta_{15} + \frac{1}{b} + 1\right)\delta_f,$$

for which a simple numerical analysis shows that the maximum value is slightly below $3.05\delta_f$.

We complete the proof by showing that $E_{\theta,f}(b) < E_{3,15}(b)$ for $f > 15$, with θ defined by (6). Since θ is increasing as a function of f, let f_θ be the lowest value of f such that $\theta = \left\lceil \log_2 \log_\alpha \delta_f \right\rceil$. Then, f_θ is also increasing as a function of θ.

Since it is clear that $E_{\theta,f_\theta} > E_{\theta,f}$ for all $f > f_\theta$, it suffices to bound E_{θ,f_θ}. To that end, we will consider the three terms in the definition of $E_{\theta,f}$ that depend on θ and f separately. The first term $b^{2^\theta-1}$ needs no complicated assessment. Clearly, with $0.5 \le b < 1$, this term decreases rapidly with θ.

The second term is $2b^{2^{\theta-1}}T_\theta\sqrt{\delta_f}$. Using the definition of θ, we can rewrite $\sqrt{\delta_f}$ as

$$\log_2 \log_\alpha \delta_f + \gamma = \theta \quad \Rightarrow \quad \sqrt{\delta_f} = \alpha^{2^{\theta-\gamma-1}},$$

where the value of γ depends on f and is determined by the ceiling operation. In any case, $0 \leq \gamma < 1$, taking the derivative

$$\begin{aligned}\frac{\mathrm{d}\left(2b^{2^{\theta-1}}T_\theta \alpha^{2^{\theta-\gamma-1}}\right)}{\mathrm{d}\theta} &= 2b^{2^{\theta-1}}\alpha^{2^{\theta-\gamma-1}}\left(\tfrac{1}{b}+1+T_\theta 2^{\theta-1}\ln 2(\ln b + 2^{-\gamma}\ln\alpha)\right)\\ &< 6b^{2^{\theta-1}}\alpha^{2^{\theta-\gamma-1}}\left(1+(\theta-1)2^{\theta-2}\ln 2\ln\alpha\right),\end{aligned}$$

where we used $c_0(b) < 2$ and $\mathrm{d}T_\theta/\mathrm{d}\theta < 3$, so that $T_\theta < 3(\theta-1)$. The factor before the outer parentheses is positive for any valid b and $\theta \geq 3$. With initial approximation (2), $\alpha = 3/2 - \sqrt{2} \approx 0.085786$, and it is easy to verify that the factor between the outer parentheses is negative for $\theta = 3$. Moreover, the negative part will only increase in (absolute) size with θ. Therefore, the derivative is, and will remain, negative. This shows that the original term is decreasing as a function of θ.

The third term is $bT_\theta^2\delta_f$. Similarly, writing δ_f as a function of θ and taking the derivative, we find

$$\begin{aligned}\frac{\mathrm{d}\left(bT_\theta^2\alpha^{2^{\theta-\gamma}}\right)}{\mathrm{d}\theta} &= bT_\theta\alpha^{2^{\theta-\gamma}}\left(2(\tfrac{1}{b}+1)+T_\theta 2^{\theta-\gamma}\ln 2\ln\alpha\right)\\ &< 6bT_\theta\alpha^{2^{\theta-\gamma}}\left(1+(\theta-1)2^{\theta-2}\ln 2\ln\alpha\right).\end{aligned}$$

This resembles the bound for the second term. Indeed, the term before the outer parentheses is again positive for any valid b and $\theta \geq 3$, and with the known value for α it is easy to verify that the factor between the outer parentheses is negative for $\theta = 3$. Moreover, the negative part will only increase in (absolute) size with θ. Therefore, the derivative is always negative, which shows that the original term is decreasing as a function of θ.

Combining these results shows that $E_{\theta,f}(b) < E_{3,15}(b) < 3.05$, for all $b \in [0.5, 1)$ and $f > 15$, which proves the statement. Note that we could tighten the bound even more by computing E_{θ,f_θ} for an arbitrary $\theta > 3$. □

To limit the absolute error for the computation of $1/a$ to $\delta_f = 2^{-f}$, we apply Algorithm 2 using n additional bits of precision. That is, we use fixed-point arithmetic with $f+n$ fractional bits in the core of Algorithm 2. The downside of using extra bits is that more bits need to be truncated after every multiplication, and secure truncation is a relatively expensive procedure. Notice that we may still directly apply Theorem 1 to find that $|\epsilon_{\theta,n}| < \rho\delta_{f+n}$. After finishing the Newton–Raphson iterations, the result should be rounded to the original precision, which may introduce more errors. We will evaluate these errors in the next section, together with the errors introduced in the scaling steps.

3.3. Tight Error Analysis

In this section, we analyze the errors due to the scaling steps in Algorithm 2. The input a is scaled to $b = av \in [0.5, 1]$, and the output is obtained by scaling $c_\theta \approx 1/b$ to $d_\theta = c_\theta v \approx 1/a$. The scaling steps introduce additional errors or magnify existing errors. Up until this point, we silently assumed that the scaling $b = av$ was exact. This, however, may not be true if $|a| > 1$. In this case, the radix point shifts to the left, and because we are working with fixed-point numbers, the least significant bits are lost. So, instead of $b = av$, we obtain

$$b^* = \lfloor av \rceil_\$ = av + \eta_1,$$

where η_1 is the error induced by the scaling from a to b. An important observation is that $|\eta_1|$ is smaller than the precision used in the computation. Moreover, b^* can be computed with the same number of fractional bits as the intermediate results in the Newton–Raphson

iterations (it would be a waste to scale down to f fractional bits if the computation is performed with $f+n$ fractional bits). Consequently, we know that

$$|\eta_1| < \delta_{f+n} = \frac{\delta_f}{2^n}.$$

After scaling, the reciprocal of b^* is computed. As explained at the end of Section 3.2, these computations are performed with extra bits. However, at this point we do not yet reduce the precision back to δ_f, but instead use

$$c_\theta^* = \frac{1}{b^*} - \epsilon_{\theta,n},$$

where $|\epsilon_{\theta,n}| < \rho\delta_{f+n}$. Recall that $\rho = 3.05$, according to Theorem 1. This result is scaled back through another multiplication by v and subsequently rounded deterministically to the original precision:

$$d_\theta^* = c_\theta^* v + \eta_2,$$

where $|\eta_2| \leq \frac{1}{2}\delta_f$. The absolute error then reads as:

$$\begin{aligned} |d_\theta^* - d| &= \left| \left(\frac{1}{av+\eta_1} - \epsilon_{\theta,n} \right) v + \eta_2 - \frac{1}{a} \right| \\ &= \left| \frac{1}{a} \frac{1}{1+\frac{\eta_1}{av}} - \epsilon_{\theta,n} v + \eta_2 - \frac{1}{a} \right| \\ &= \left| \frac{1}{a} \left(1 - \frac{\eta_1}{av} + \left(\frac{\eta_1}{av} \right)^2 - \dots \right) - \epsilon_{\theta,n} v + \eta_2 - \frac{1}{a} \right| \\ &= \left| \frac{1}{a} \left(\frac{-\eta_1}{b+\eta_1} \right) - \epsilon_{\theta,n} v + \eta_2 \right|. \end{aligned} \tag{8}$$

A careful analysis, partly covered by Lemmas A3 and A4, leads to the following result:

Theorem 2. *If the Newton–Raphson method is used to compute* $1/a$ *for* $a \in \mathbb{Q}_{2f,f}$*, employing the approach in Algorithm 2, with* $n \leq f$*, then the absolute error* (8) *is bounded by* 2^{-n}*.*

Proof. We distinguish three cases: (i) $a < 2^{-n}$, (ii) $2^{-n} \leq a < 2^n$, and (iii) $a \geq 2^n$. In case (i), $v \geq 2^n$ and, consequently, both scaling steps introduce no rounding errors: $\eta_1 = \eta_2 = 0$. In case (ii), $2^{-n} \leq v \leq 2^{n-1}$. Due to the extra precision that is used, there is still no rounding error in the initial scaling step ($\eta_1 = 0$), but there might be an error when the result is truncated to the original precision. In case (iii), $v \leq 2^{-(n+1)}$, and both scaling steps may introduce errors.

In case (i), the absolute error simplifies to $|-\epsilon_{\theta,n} v|$. For such small values of a, the scaling factor v is large, and the absolute error is bounded by $2^{f-1}\rho\delta_{f+n}$. However, it can be shown to be tighter by noting that $v = 2^{f-1}$ occurs only when $a = \delta_f$. For this value of a, according to Lemma A3, ρ may be replaced by 2. For the remaining values of a in case (i), $\epsilon_{\theta,n}$ may be approximately 1.5 times as large (it is still bounded by $\rho\delta_{f+n}$), but the value for v is at most 2^{f-2}, making the product $|-\epsilon_{\theta,n} v|$ strictly smaller. Thus, the exact bound for case (i) is $2^{f-1} 2\delta_{f+n} = 2^{-n}$.

In case (ii), the error simplifies to $|-\epsilon_{\theta,n} v + \eta_2|$, which is bounded by $2^{n-1}\rho\delta_{f+n} + \frac{1}{2}\delta_f = \frac{1}{2}(\rho+1)\delta_f$. Using the value for ρ given in Theorem 1, it is straightforward to deduce that the bound for case (i) exceeds that of case (ii) if $n \leq f-2$:

$$2^{-n} \geq 2^{-(f-2)} = 4\delta_f > \frac{\rho+1}{2}\delta_f.$$

The cases $n = f-1$ and $n = f$ are less straightforward and will be considered separately.

If $n = f - 1$, case (i) contains only $a = \delta_f$. As already derived, the error in this case is bounded by $2^{-n} = 2\delta_f$. The first value in case (ii) is $2\delta_f$, for which we may replace $\rho\delta_{f+n}$ by $2\delta_{f+n}$, according to Lemma A3. The maximal error before applying η_2 then reads $2^{n-1}2\delta_{f+n} = \delta_f$. With $a = 2\delta_f$, $1/a = 2^{f-1}$, which is a multiple of δ_f. Consequently, the error cannot become larger than $\delta_f < 2^{-n}$. The next value in case (ii) is $3\delta_f$, for which we may replace $\rho\delta_{f+n}$ by $\frac{7}{3}\delta_{f+n}$, according to Lemma A4. The maximal error before applying η_2 then reads $2^{n-1}\frac{7}{3}\delta_{f+n} = \frac{7}{6}\delta_f$. With $a = 3\delta_f$, $1/a = \frac{1}{3}2^f$, which is a multiple of $\frac{1}{3}\delta_f$ (but not an integer multiple of δ_f). Combining these results shows that the total error is bounded by $\frac{5}{3}\delta_f < 2^{-n}$. For larger values of a, the error is simply bounded by $2^{n-2}\rho\delta_{f+n} + \frac{1}{2}\delta_f = 1.2625\delta_f < 2^{-n}$.

If $n = f$, case (i) ceases to exist. The first value in case (ii) is δ_f. Similar to the case $a = 2\delta_f$ when $n = f - 1$, we know that in this case the maximal error is $\delta_f = 2^{-n}$. The next value in case (ii) is $2\delta_f$, for which a similar derivation shows that the error is bounded by $\frac{1}{2}\delta_f < 2^{-n}$. The third value in case (ii) is $3\delta_f$. Analogous to the situation with $n = f - 1$, we may replace ρ by $\frac{7}{3}\delta_{f+n}$ and find that the error before applying η_2 is bounded by $2^{n-2}\frac{7}{3}\delta_{f+n} = \frac{7}{12}\delta_f$. Knowing that the exact solution is a multiple of $\frac{1}{3}\delta_f$ (but not an integer multiple of δ_f), we conclude that the total error, after applying η_2 (deterministically), is maximally $\frac{2}{3}\delta_f < \delta_f = 2^{-n}$. Again, for larger values of a, the error is bounded by $2^{n-3}\rho\delta_{f+n} + \frac{1}{2}\delta_f = 0.88125\delta_f < 2^{-n}$.

Thus, for all $n \leq f$, the errors in cases (i) and (ii) are bounded by 2^{-n}. For even larger values of a, the error bound decreases rapidly, despite η_1 coming into play. In case (iii), the error is approximately bounded by $\frac{1}{2}(2^{-2n+2} + 2^{-2n}\rho + 1)\delta_f$ (ignoring the η_1 term in the denominator that is small compared to b), which is significantly smaller than 2^{-n}. □

Remark 4. *Concerning the relative error, given by the expression*

$$\left|\frac{d_\theta^* - d}{d}\right| = \left|\frac{-\eta_1}{b + \eta_1} - \epsilon_{\theta,n} b + a\eta_2\right|,$$

we see that the tables have turned. For small values of a, the relative error is also small, while for larger values of a the error increases. If $a < 2^{-n}$, the error is bounded by $2^{-n}\rho\delta_f$, while the bound increases to $(2^{-n}\rho + 2^{n-1})\delta_f$ for $2^{-n} \leq a < 2^n$. For larger values of a, the last term on the right-hand side starts to dominate. In this domain, the error is bounded by $(1/(2^n b - \delta_f) + 2^{-n}\rho + \frac{1}{2}(2^f - 1))\delta_f < (2^{-n+1} + 2^{-n}\rho + 2^{f-1})\delta_f \approx 0.5$. Based on the results from numerical experiments, we suspect that the actual bound for the relative error lies at approximately $1/3$, due to the relation between a and η_2 (they do not attain their maximal values at the same time). Though this error may seem large, it is not an effect of the specific computational algorithms, but merely a behavior inherent to the use of fixed-point numbers.

Corollary 1. *If the Newton–Raphson method is used to compute $1/a$ for $a \in \mathbb{Q}_{2f,f}$, using the approach in Algorithm 2, then computing with $n = f$ additional bits guarantees that the absolute error (8) is bounded by δ_f, while using $n = f + 1$ bits guarantees that the absolute error is strictly smaller than δ_f.*

Proof. According to Theorem 2, if $n \leq f$, the absolute error is bounded by 2^{-n}. It follows directly that if $n = f$, the bound equals $2^{-f} = \delta_f$. Note that from the proof of Theorem 2, it follows that this bound can only be attained when $a = \delta_f$.

If $n = f + 1$, then a proof similar to that of Theorem 2 shows that the error is strictly smaller than δ_f. Recall that for small values of a, the error reads as $|-\epsilon_{\theta,n} v + \eta_2|$, and therefore the absolute error before applying η_2 is bounded by $2^{n-2}\rho\delta_{f+n}$. For $a = \delta_f$, however, we may replace ρ by $-\delta_{f+n}$ or $2\delta_{f+n}$, according to Lemma A3. This leads to errors $-2^{n-2}\delta_{f+n} = -\frac{1}{4}\delta_f$ and $2^{n-2}2\delta_{f+n} = \frac{1}{2}\delta_f$, respectively. Because η_2 is applied deterministically, both will be rounded to the analytical solution, and hence $\epsilon_\theta = 0$.

For larger values of a, the error is bounded by $2^{n-3}\rho\delta_{f+n} + \frac{1}{2}\delta_f = 0.88125\delta_f < \delta_f$, which completes the proof. By considering the cases $a = 2\delta_f$ and $a = 3\delta_f$ separately from even larger values of a, it is possible to show that the error is actually strictly smaller than $0.7\delta_f$, but we will omit the proof here. □

4. Integer Division

Secure integer division is an important primitive and appears in many applications. For integer inputs $[\![g]\!]$, $[\![a]\!]$, performing integer division yields integers $[\![q]\!]$, $[\![r]\!]$ such that $g = qa + r$ and $0 \leq r < a$. Formulated differently, we have $q = \lfloor g/a \rfloor$ and $r = g - qa$.

One possible way of computing $[\![q]\!]$ is by applying the Newton–Raphson algorithm described in the previous section. To that end, $[\![a]\!]$ needs to be converted from an integer to a fixed-point number. Subsequently, the reciprocal of $[\![a]\!]$ is computed and multiplied by $[\![g]\!]$. It turns out, however, that it is advantageous to perform the multiplication by $[\![g]\!]$ before finalizing the computation of $1/[\![a]\!]$. The resulting value $[\![\tilde{q}]\!]$ is a good approximation to $[\![q]\!]$ and can be used to compute the final, correct value of $[\![q]\!]$. In the remainder of this section, we will omit the secret-shared brackets for better readability.

4.1. Error for Integer Division

In the case of integer division, the error analysis from Section 3.3 can be simplified. With a being an integer, there can only be nonzero bits to the left of the radix point. This means—assuming that there are an equal number of bits before and after the radix point, i.e., $\ell = 2f$—that no information is lost in the initial scaling step: $\eta_1 = 0$. In other words, in the case of integer division, we have $b^* = b$. The error after computing the reciprocal of b (before rescaling and truncating to the original precision) is still bounded by $\epsilon_{\theta,n}$, such that we now have $c_\theta = 1/(av) - \epsilon_{\theta,n}$.

At this point, we first multiply g and v. Since we have assumed that $\ell = 2f$, there is no rounding error for this multiplication. The result is multiplied by c_θ, after which we truncate to the original precision. This results in a generally nonintegral estimate to g/a, which we call $\tilde{q}$:

$$\begin{aligned} \tilde{q} &= \left(\frac{1}{av} - \epsilon_{\theta,n}\right) gv + \eta_2 \\ &= \frac{g}{a} - \epsilon_{\theta,n} gv + \eta_2. \end{aligned}$$

The resulting approach is summarized in Algorithm 3. In what follows, we will denote the error of $\tilde{q}$ (with respect to g/a) by $E_{\tilde{q}}$.

Algorithm 3 IntDivFxp($[\![g]\!], [\![a]\!], n = 1$) $-2^{\ell-1} \leq g, a < 2^{\ell-1}$, with $g, a \in 2^f\mathbb{Z}$

Lines 1–8 of Algorithm 2
Line 2 simplifies to $[\![b]\!] \leftarrow 2^{-f+n}[\![a]\!][\![v]\!]$
9: $[\![w]\!] \leftarrow 2^{-f}[\![g]\!][\![v]\!]$
10: $[\![\tilde{q}]\!] \leftarrow \mathsf{Round}_{f+n}([\![c_\theta]\!][\![w]\!])$
11: **return** $[\![\tilde{q}]\!]$ ▷ $-2^{\ell-1} \leq \tilde{q} < 2^{\ell-1}$

Theorem 3. *If the Newton–Raphson method is used to compute g/a, with g and a being integers, using the approach in Algorithm 3 and $n < f$, then $|E_{\tilde{q}}| \leq 2^{-n}$.*

Proof. To derive the error bound, we consider the error before the final truncation η_2 is applied: $-\epsilon_{\theta,n} gv$. Because a is an integer, we have $v \leq 0.5$, with equality only when $a = 1$. In the latter case, $b = 0.5$, and according to Lemma A3 we have $|-\epsilon_{\theta,n}(0.5)| \leq 2\delta_{f+n}$. It

follows that $|-\epsilon_{\theta,n}v| \leq \delta_{f+n}$. Furthermore, we know that $g \leq 2^f - 1$. Combining all this gives

$$\begin{aligned} |-\epsilon_{\theta,n}gv| &\leq \delta_{f+n}(2^f-1) \\ &= (1-2^{-f})2^{-n} \\ &< 2^{-n}. \end{aligned}$$

Obviously, when $a = 1$, g/a is an integer, and therefore a multiple of δ_f. Since $n < f$, 2^{-n} is also a multiple of δ_f. From these observations, it follows that the final rounding η_2 cannot bring the error any further than 2^{-n}.

The case $v = 0.25$ occurs only for $a = 2$ and $a = 3$, leading to $b = 0.5$ and $b = 0.75$, respectively. Clearly, for $a = 2$, we have again that $|\epsilon_{\theta,n}(0.5)| \leq 2\delta_{f+n}$, leading to $|-\epsilon_{\theta,n}gv| < \frac{1}{2}2^{-n}$. Because $|\eta_2| < \delta_f \leq \frac{1}{2}2^{-n}$, we know that $|-\epsilon_{\theta,n}gv + \eta_2| < 2^{-n}$. For the case $a = 3$, let us write $n = f - \gamma$, so that $2^{-n} = 2^{\gamma}\delta_f$ ($\gamma = 1, 2, 3, \ldots$). According to Lemma A4, we have $|\epsilon_{\theta,n}(0.75)| \leq \frac{7}{3}\delta_{f+n}$ when $n + f = 6$, leading to $|-\epsilon_{\theta,n}gv| < \frac{7}{12}2^{-n}$. The final error can only be larger than 2^{-n} when $|\eta_2| > \frac{5}{12}2^{-n} = \frac{5}{12}2^{\gamma}\delta_f$, and because $|\eta_2| < \delta_f$, this is only possible if $\gamma = 1$. However, the system $n + f = 6$ and $n = f - 1$ has no integer solutions, and therefore this scenario will never occur. Lemma A4 tells us that in all other cases $|-\epsilon_{\theta,n}(0.75)| \leq \frac{5}{3}\delta_{f+n}$, leading to $|\epsilon_{\theta,n}gv| < \frac{5}{12}2^{-n}$. Now, the final error can only be larger than 2^{-n} when $|\eta_2| > \frac{7}{12}2^{-n} = \frac{7}{12}2^{\gamma}\delta_f$. Because $|\eta_2| < \delta_f$ and $\gamma \geq 1$, this is impossible.

For even larger values of a, $v \leq 0.125$, and we have $|\epsilon_{\theta,n}| \leq \rho\delta_{f+n}$, leading to $|-\epsilon_{\theta,n}gv| < \frac{1}{8}\rho 2^{-n}$. The final error can only be larger than 2^{-n} if $|\eta_2| > \frac{7}{8}\rho 2^{-n} = \frac{7}{8}\rho 2^{\gamma}\delta_f$. Again, because $|\eta_2| < \delta_f$, there are no solutions with $\gamma \geq 1$. □

We emphasize that the above result holds even when η_2 is determined probabilistically, whereas throughout Section 3 it was assumed that the final rounding—to the original precision—was performed deterministically (which was especially relevant for the cases $n = f$ and $n = f + 1$).

4.2. From Fixed-Point Approximation to Integer Solution

The fixed-point value $\tilde{q}$ now needs to be rounded to an integer value $\bar{q}$. This can be achieved either deterministically or probabilistically.

Corollary 2. *Suppose $\tilde{q}$ is computed with Algorithm 3 using $n = 1$. If $\tilde{q}$ is rounded to an integer $\bar{q}$ deterministically, then $\bar{q} \in \{q, q+1\}$. If $\tilde{q}$ is rounded to an integer $\bar{q}$ probabilistically, then $\bar{q} \in \{q-1, q, q+1\}$.*

Proof. We apply Theorem 3 to find that the error on $\tilde{q}$ is bounded by 2^{-1}. Since $q \leq g/a < q + 1$, this gives $q - 0.5 \leq \tilde{q} < (q + 1) + 0.5$. It follows directly that for deterministic rounding, $\bar{q} \in \{q, q+1\}$. It also follows that for probabilistic rounding, $\bar{q} \in \{q-1, q, q+1, q+2\}$. It remains to be shown that $\bar{q} = q + 2$ is not possible. To that end, first note that from $q \leq g/a < q + 1$, it follows that $q \leq g/a \leq q + 1 - 1/a$. Therefore, for $\bar{q} = q + 2$ to occur, it should be possible that $E_{\tilde{q}} > 1/a$.

Suppose that $2^{m-1} \leq a < 2^m$ for some integer m. Then, $v = 2^{-m}$ and $1/a = v/b$. At this point, we are only interested in solutions $\tilde{q} > g/a$ with negative errors, hence we have $|\epsilon_{\theta,n}| < (1/b + 1)\delta_{f+n}$. Maximizing the error $|-\epsilon_{\theta,n}gv|$ with $n = 1$ then gives

$$\begin{aligned} |-\epsilon_{\theta,1}gv| &\leq (1/b+1)\delta_{f+1}(2^f-1)v \\ &= \tfrac{1}{2}(1/b+1)(1-2^{-f})v \\ &< \tfrac{1}{2}(1/b+1)v \\ &\leq v/b \\ &= 1/a. \end{aligned}$$

Thus, the approximation before applying η_2 is still below $q+1$. Since $q+1$ is an integer and therefore a multiple of δ_f, it follows that $|E_{\tilde{q}}| \leq 1/a$. In other words, the rounding η_2 cannot push the error beyond $q+1$. Consequently, $\tilde{q}$ will never be rounded to $q+2$. □

Remark 5. *If we were to calculate $\tilde{q}$ with Algorithm 2 instead of Algorithm 3 and multiply the result by g, we would find that*

$$\begin{aligned} \tilde{q} &= \left(\left(\frac{1}{av} - \epsilon_{\theta,n}\right)v + \eta_2\right)g \\ &= \frac{g}{a} - \epsilon_{\theta,n}gv + \eta_2 g. \end{aligned}$$

Numerical simulations suggest that we would then find the same values for $\bar{q}$. That is, $\bar{q} \in \{q, q+1\}$ in the case of deterministic rounding and $\bar{q} \in \{q-1, q, q+1\}$ in the case of probabilistic rounding. However, this approach would require an extra secure comparison in the deterministic rounding step in line 9 of Algorithm 2.

If we were to replace this deterministic rounding with probabilistic rounding, then $|\eta_2| < \delta_f$ (instead of $|\eta_2| \leq \frac{1}{2}\delta_f$ with deterministic rounding). Numerical simulations show that in this case, $\bar{q} \in \{q-1, q, q+1, q+2\}$, independent of whether rounding to an integer is performed deterministically or probabilistically. Hence, in this approach, at least one extra secure comparison is also required to find the correct value q. This proves that it is indeed advantageous to incorporate multiplication by g into the computation of $1/a$, as we did in Algorithm 3.

So far, we have computed $\tilde{q}$ using only probabilistic rounding. We found that $\bar{q} \in \{q, q+1\}$ if the rounding (to the nearest) is performed deterministically and $\bar{q} \in \{q-1, q, q+1\}$ if $\tilde{q}$ is rounded to $\bar{q}$ probabilistically. The final step is to recover the correct solution q.

This is achieved by one or two comparisons, depending on how $\tilde{q}$ is rounded to $\bar{q}$. According to Corollary 2, if $\tilde{q}$ is rounded deterministically, then $\bar{q} \in \{q, q+1\}$. Hence, we can compute $\bar{q}a - g$ and check the sign. If $\bar{q}a - g > 0$, then $q = \bar{q} - 1$; otherwise, $q = \bar{q}$. If $\tilde{q}$ is rounded probabilistically, then $\bar{q} \in \{q-1, q, q+1\}$. This time, we not only check the sign of $\bar{q}a - g$, but also that of $(\bar{q}+1)a - g$. If $\bar{q}a - g > 0$, then $q = \bar{q} - 1$. Otherwise, if $(\bar{q}+1)a - g > 0$, then $q = \bar{q}$, or $q = \bar{q} + 1$.

At first sight, it might not seem relevant if $\tilde{q}$ is rounded to $\bar{q}$ deterministically or probabilistically, because even though deterministic rounding requires an extra secure comparison, it saves a secure comparison in the computation of q. Rounding probabilistically to $\bar{q}$ does not require any secure comparisons, but two secure comparisons are needed to find the correct value for q. Hence, in both cases, we need exactly two secure comparisons. However, the secure comparison in Algorithm 1 is cheaper than a regular secure comparison, because the bits of the numbers that are compared are already available. Therefore, it is computationally advantageous to choose the option with deterministic rounding to $\bar{q}$ and only one comparison to find q. The complete procedure is summarized in Algorithm 4.

Algorithm 4 IntDiv($[\![g]\!], [\![a]\!]$) $-2^{f-1} \leq g, a < 2^{f-1}$, with $g, a \in \mathbb{Z}$

1: $[\![\tilde{q}]\!] \leftarrow$ IntDivFxp($[\![g2^f]\!], [\![a2^f]\!]$) ▷ $-2^{\ell-1} \leq \tilde{q} < 2^{\ell-1}$
2: $[\![\bar{q}]\!] \leftarrow$ Round$_f$($[\![\tilde{q}]\!]$, deterministic)
3: $[\![q]\!] = [\![\bar{q}]\!] - ([\![\bar{q}]\!][\![a]\!] > [\![g]\!])$
4: **return** $[\![q]\!]$ ▷ $-2^{f-1} \leq q < 2^{f-1}$

5. Reciprocal Square Root

To compute the reciprocal (or, inverse) square root securely, we follow the same approach as in Section 3 for the reciprocal. The overall goal is to guarantee an absolute error not exceeding $\delta_f = 2^{-f}$ while minimizing the additional precision used during the

computation. In Section 6, we will use this result for the secure computation of the square root with the same accuracy.

5.1. Secure Computation

The reciprocal square root function evaluates $1/\sqrt{[\![a]\!]}$ for a secret-shared value $[\![a]\!]$, $a > 0$, see Algorithm 5. Upon initialization, $[\![a]\!]$ is scaled to $[\![b]\!] = [\![a]\!][\![v]\!]$ such that $b \in [0.5, 2)$. The interval for b is taken twice as large as that for the reciprocal, so that the scaling factor $v = 2^k$, $k \in \mathbb{Z}$, can be chosen with k even. This ensures that scaling back by $[\![v^{1/2}]\!]$ at the end introduces no additional rounding errors.

Algorithm 5 RecSqrt($[\![a]\!], n = 0$) $\quad -2^{\ell-1} \le a < 2^{\ell-1}$

1: $[\![v]\!], [\![v^{\frac{1}{2}}]\!] \leftarrow \mathsf{Scale}([\![a]\!])$ ▷ $v = \pm 2^k, k \in \mathbb{Z}$, k even
2: $[\![b]\!] \leftarrow \mathsf{Round}_{f-n}([\![a]\!][\![v]\!])$ ▷ $2^{f+n-1} \le b < 2^{f+n+1}$
3: $\beta \leftarrow (\sqrt{2}-1)/4$
4: $\tau \leftarrow 3/\sqrt{2}$
5: $\theta \leftarrow \lceil \log_2 \log_{\tau\beta}(\tau 2^{-(f+n)}) \rceil$
6: $[\![c_0]\!] \leftarrow 3/2 + \beta - \mathsf{Round}_1([\![b]\!]/2, \text{deterministic})$
7: **for** $i = 1$ **to** θ **do**
8: $\quad [\![z_1]\!] \leftarrow \mathsf{Round}_{f+n}([\![c_{i-1}]\!][\![b]\!]))$
9: $\quad [\![z_2]\!] \leftarrow 3 - \mathsf{Round}_{f+n}([\![c_{i-1}]\!][\![z_1]\!])$
10: $\quad [\![c_i]\!] \leftarrow \mathsf{Round}_{f+n+1}(\frac{1}{2}[\![c_{i-1}]\!][\![z_2]\!])$
11: $[\![d_\theta]\!] \leftarrow \mathsf{Round}_{f+n}([\![c_\theta]\!][\![v^{\frac{1}{2}}]\!], \text{deterministic})$
12: **return** $[\![d_\theta]\!]$ ▷ $-2^{\ell-1} \le d_\theta < 2^{\ell-1}$

To find an initial approximation, following the same approach that led to (2) would give

$$[\![c_0]\!] = \tfrac{\sqrt{2}}{6}(7 - 2[\![b]\!]) - \alpha^*,$$

where $\alpha^* = (7 - 3\sqrt[3]{9})/(6\sqrt{2})$. This initial approximation has a maximal absolute error of $\alpha^* \approx 0.089537$ (at $b = 0.5$, $b = \sqrt[3]{9}/2$, and $b = 2$). An integer factor in front of b—like in (2)—would be more efficient, but this is not really an option here. A factor $\frac{1}{2}$ is possible, essentially reducing the cost of truncation by a factor of f. Therefore, another good initial approximation is

$$[\![c_0]\!] = \tfrac{1}{4}(5 + \sqrt{2}) - \tfrac{1}{2}[\![b]\!], \tag{9}$$

which has a maximal absolute error of $\beta = \frac{1}{4}(\sqrt{2}-1) \approx 0.103553$ at $b = 1$ and $b = 2$ and only $\frac{1}{4}(3\sqrt{2}-4) \approx 0.060660$ at $b = 0.5$. Obviously, this slightly higher initial error may lead to an extra iteration in some cases, but it turns out this is not the case for the most common values of f, namely $f = 2^n$ with $n \in \{2, \ldots, 10\}$. Compared to the initial approximation by Liedel [4], our approximation is slightly less accurate. This may be attributed to the fact that the approximation by Liedel was derived for the interval $[0.5, 1)$, while ours is defined for $[0.5, 2)$. Due to the quadratic convergence behavior of the Newton–Raphson method, however, the effect of the lower initial accuracy is rather small. On the other hand, our approximation is more efficient in terms of truncation, because (a) we only need to truncate a single bit to compute c_0, whereas Liedel needed many more, and (b) Liedel assumed that the input is scaled to $[0.5, 1)$, so it is possible that the square root of the scaling factor is not an integral power of two. In these cases, another multiplication by $\sqrt{2}$ needs to be performed, leading to another expensive truncation. Because our approximation and method rely on the assumption that the input is scaled to $[0.5, 2)$ in such a way that the scaling factor is always an even power of two, no such correcting multiplication is necessary. Aly and Smart [5] used an even more crude initial approximation. It required the position of the most significant bit, say t, which was then used to compute $2^{t/2}$, a rough approximation for the square root. Finding the most

significant bit, however, is equivalent to our scaling step to find b, and once b is known, computing the more accurate approximation in (9) is basically free.

Given the initial approximation $[\![c_0]\!]$, successive approximations are computed using

$$[\![c_{i+1}]\!] = \tfrac{1}{2}[\![c_i]\!]\Big(3 - [\![c_i]\!]^2[\![b]\!]\Big), \tag{10}$$

which corresponds to the Newton–Raphson method in (1) applied to $f(c) = b - 1/c^2$. After θ iterations, with θ independent of the input value, the scaling is inverted. The final approximation for $1/\sqrt{a}$ then reads

$$[\![d_\theta]\!] = [\![c_\theta]\!][\![v^{\frac{1}{2}}]\!].$$

Now, we clearly see why the scaling factor v was chosen to be an even power of two: it also makes the inverse scaling an integral power of two.

The required number of iterations θ will be determined below such that the final error for c_θ does not exceed $\delta_f = 2^{-f}$, assuming exact arithmetic. Subsequently, we will determine the required number of additional bits n for Algorithm 5, taking into account all (rounding) errors. For better readability, we will drop the secret-shared brackets in the remainder of this section.

Using $c = 1/\sqrt{b}$ to denote the analytical solution and $\epsilon_i = c - c_i$ to denote the iteration error, applying (10) gives

$$\epsilon_{i+1} = c - \tfrac{1}{2}c_i(3 - c_i^2 b) = c - \tfrac{1}{2}(c - \epsilon_i)(3 - (c - \epsilon_i)^2)b) = \tfrac{3}{2}\sqrt{b}\epsilon_i^2 - \tfrac{1}{2}b\epsilon_i^3. \tag{11}$$

Since $b \in [0.5,\ 2)$ and $|\epsilon_0| \leq \beta < 1$, we see that quadratic convergence is guaranteed right from the start. From (11), it follows directly that for those values of b where $\epsilon_0 \geq 0$, it holds that

$$|\epsilon_i| \leq (\tfrac{3}{2}\sqrt{b})^{2^i-1}\epsilon_0^{2^i} \leq (\tau\beta)^{2^i}/\tau,$$

where $\tau = 3/\sqrt{2}$ and the convergence is slowest for $b = 2$. To achieve $|\epsilon_\theta| \leq \delta_f$, we thus set

$$\theta = \lceil \log_2 \log_{\tau\beta}(\tau\delta_f) \rceil. \tag{12}$$

For those values of b where $\epsilon_0 < 0$, the third-order term in (11) cannot simply be ignored. However, we will not update (12) accordingly. Instead, we will handle these cases in the appropriate places in the proofs that follow.

5.2. Tight Error Analysis without Scaling

In this section, we analyze the error $\epsilon_\theta = c - c_\theta$ in the computation of $c = 1/\sqrt{b}$ for $b \in [0.5,\ 2)$. Analogous to the analysis for the reciprocal, we determine a tight bound for $|\epsilon_\theta|$ taking into account all (rounding) errors, assuming fixed-point arithmetic with f fractional bits in Algorithm 5 (thus with $n = 0$). In Section 5.3, we will use this bound to determine the minimal number of additional bits n needed to guarantee that the absolute error for $1/\sqrt{a}$ is limited to $\delta_f = 2^{-f}$, also taking into account the errors due to scaling.

With the help of Lemmas A5 and A6, we are able to give a bound on the total error for the reciprocal square root after θ iterations.

Theorem 4. *If the Newton–Raphson method is used to compute* $1/\sqrt{b}$ *for some* $b \in [0.5, 2)$*, employing initial approximation* (9) *and with the number of iterations* θ *computed via* (12)*, then* $|\epsilon_\theta| < \sigma\delta_f$*, where* $\sigma = 2.71$*.*

Proof. Clearly, if $\theta = 0$, the initial error is already below δ_f. Because no iterations are performed, no further errors are introduced, and the final error remains below δ_f.

For the cases in which $\theta \in \{1,2,3\}$, we exhaustively compute the error for all possible inputs b, taking into account all rounding possibilities. This covers the values $4 \leq f \leq 18$ and yields a maximum value for $|\epsilon_\theta|$ of approximately $2.60\delta_f$.

For larger values of f, we follow an analogous approach to that for the reciprocal: firstly, we derived an expression that bounds the absolute error as a function of f and θ. Secondly, we compute the value of the error bound for $f = 19$ ($\theta = 4$), which will be below $2.71\delta_f$. Thirdly, we show that for larger values of f, the value of the error bound will always be smaller than in the case $f = 19$.

Following Lemma A5, we know that in the case of exact arithmetic, the error at the start of the final iteration is bounded by $\xi^{2^{\theta-2}}\sqrt{b/2}^{2^{\theta-1}-1}\sqrt{\delta_f/\tau}$. Lemma A6 tells us that in the first iteration, the rounding error is bounded by $(c_0^2/2 + c_0/2 + 1)\delta_f$, while in every subsequent iteration it is bounded by $(1/(2b) + 1/(2\sqrt{b}) + 1)\delta_f$. Thus, for $\theta \geq 4$, we obtain the following bound for the total error at the start of the final iteration:

$$|\epsilon_{\theta-1}| < \xi^{2^{\theta-2}}\sqrt{b/2}^{2^{\theta-1}-1}\sqrt{\delta_f/\tau} + \left(\frac{c_0^2}{2} + \frac{c_0}{2} + 1\right)\delta_f + (\theta-2)\left(\frac{1}{2b} + \frac{1}{2\sqrt{b}} + 1\right)\delta_f.$$

Let $T_\theta = T_\theta(b) = (c_0^2/2 + c_0/2 + 1) + (\theta-2)(1/(2b) + 1/(2\sqrt{b}) + 1)$. Applying (A4) with $i = \theta - 1$, and without the third-order term (since $\epsilon_{\theta-1} > 0$), gives

$$\begin{aligned}
|\epsilon_\theta| &< \tfrac{3}{2}\sqrt{b}\epsilon_{\theta-1}^2 + \left(\frac{1}{2b} + \frac{1}{2\sqrt{b}} + 1\right)\delta_f \\
&< \tfrac{3}{2}\sqrt{b}\left(\xi^{2^{\theta-2}}\sqrt{b/2}^{2^{\theta-1}-1}\sqrt{\delta_f/\tau} + T_\theta\delta_f\right)^2 + \left(\frac{1}{2b} + \frac{1}{2\sqrt{b}} + 1\right)\delta_f \\
&= \left(\sqrt{\xi}\sqrt{b\xi/2}^{2^\theta-1} + 2\sqrt{b\xi/2}^{2^{\theta-1}}T_\theta\sqrt{\tau\delta_f} + \tfrac{3}{2}\sqrt{b}T_\theta^2\delta_f + \frac{1}{2b} + \frac{1}{2\sqrt{b}} + 1\right)\delta_f \\
&\stackrel{\text{def}}{=} E_{\theta,f}(b)\,\delta_f.
\end{aligned}$$

For the case $f = 19$, where $\theta = 4$, this yields

$$E_{4,19}(b)\,\delta_f = \left(\sqrt{\xi}\sqrt{b\xi/2}^{15} + 2\sqrt{b\xi/2}^{8}T_4\sqrt{\tau\delta_{19}} + \tfrac{3}{2}\sqrt{b}T_4^2\delta_{19} + \frac{1}{2b} + \frac{1}{2\sqrt{b}} + 1\right)\delta_f,$$

for which a simple numerical analysis shows that the maximum value is slightly below $2.71\delta_f$.

We complete the proof by showing that $E_{\theta,f}(b) < E_{4,19}(b)$ for $f > 19$, with θ defined by (12). Since θ is increasing as a function of f, let f_θ be the lowest value of f such that $\theta = \lceil \log_2 \log_{\tau\beta}(\tau\delta_f) \rceil$. Then, f_θ is also increasing as a function of θ.

Since it is clear that $E_{\theta,f_\theta} > E_{\theta,f}$ for all $f > f_\theta$, it suffices to bound E_{θ,f_θ}. To that end, we will consider the three terms in the definition of $E_{\theta,f}$ that depend on θ and f separately.

To evaluate the first term $\sqrt{\xi}\sqrt{b\xi/2}^{2^\theta-1}$, we note that ξ is defined to have the value 1.045 for $b_1 < b < b_2$, with $b_2 \approx 1.65$, while $\xi = 1$ for $b > b_2$. It thus follows that $b\xi/2 < 1$, and as a result the entire term decreases rapidly with θ.

For convenience, we use $\tilde{b} = b\xi/2$ and $\tilde{\alpha} = \tau\beta$ in the analysis below. The second term may then be written as $2\tilde{b}^{2^{\theta-2}}T_\theta\sqrt{\tau\delta_f}$. Using the definition of θ, we get $\sqrt{\tau\delta_f} = \tilde{\alpha}^{2^{\theta-\gamma-1}}$, where γ satisfies $\log_2 \log_{\tilde{\alpha}}(\tau\delta_f) + \gamma = \theta$ and $0 \leq \gamma < 1$. Taking the derivative thus yields for the second term

$$\begin{aligned}
\frac{\mathrm{d}\left(2\tilde{b}^{2^{\theta-2}}T_\theta\tilde{\alpha}^{2^{\theta-\gamma-1}}\right)}{\mathrm{d}\theta} &= 2\tilde{b}^{2^{\theta-2}}\tilde{\alpha}^{2^{\theta-\gamma-1}}\left(\tfrac{1}{2b} + \tfrac{1}{2\sqrt{b}} + 1 + T_\theta 2^{\theta-1}\ln 2(\tfrac{1}{2}\ln\tilde{b} + 2^{-\gamma}\ln\tilde{\alpha})\right) \\
&< 6\tilde{b}^{2^{\theta-2}}\tilde{\alpha}^{2^{\theta-\gamma-1}}\left(1 + (\theta-1)2^{\theta-2}\ln 2\ln\tilde{\alpha}\right),
\end{aligned}$$

where $c_0(b) < 3/2$ and $\mathrm{d}T_\theta/\mathrm{d}\theta < 3$, so that $T_\theta < 3(\theta - 1)$. This bound is almost identical to the bound we found in the analysis for the reciprocal. The factor before the outer parentheses is now positive for any valid b and $\theta \geq 4$. Additionally, it is easy to verify that the factor within the outer parentheses is negative for $\theta = 4$. Because the negative part will only increase in (absolute) size with θ, the derivative is, and will remain, negative. This shows that the original term is decreasing as a function of θ.

The third term is $\frac{3}{2}\sqrt{b}T_\theta^2\delta_f$. Writing $\tau\delta_f$ as a function of θ, we may rewrite this term to $\sqrt{b/2}T_\theta^2\tilde{\alpha}^{2^{\theta-\gamma}}$. Taking the derivative, we find:

$$\begin{aligned}\frac{\mathrm{d}\left(\sqrt{b/2}T_\theta^2\tilde{\alpha}^{2^{\theta-\gamma}}\right)}{\mathrm{d}\theta} &= 2\sqrt{b/2}T_\theta\tilde{\alpha}^{2^{\theta-\gamma}}\left(\tfrac{1}{2b} + \tfrac{1}{2\sqrt{b}} + 1 + T_\theta 2^{\theta-\gamma-1}\ln 2\ln\tilde{\alpha}\right)\\ &< 6\sqrt{b/2}T_\theta\alpha^{2^{\theta-\gamma}}\left(1 + (\theta-1)2^{\theta-2}\ln 2\ln\tilde{\alpha}\right).\end{aligned}$$

Again, this bound is very similar to the bound in the analysis for the reciprocal. The term before the outer parentheses is positive for any valid b and $\theta \geq 4$, and with the known value for $\tilde{\alpha}$ it is easy to verify that the term between the outer parentheses is negative for $\theta = 4$. Additionally, the negative part will only increase in (absolute) size with θ. Therefore, the derivative is always negative, which shows that the original term is decreasing as a function of θ.

Combining these results shows that $E_{\theta,f}(b) < E_{4,19}(b) < 2.71$ for all $b \in [0.5, 2)$ and $f > 19$, which proves the statement. Note that we could tighten the bound even more by computing E_{θ,f_θ} for an arbitrary $\theta > 4$. □

Similar to the reciprocal, we will perform our computations with extra precision to control the effect of rounding. Therefore, in the following, we assume a total of $f + n$ fractional bits. Then, we apply Theorem 4 to find that $\epsilon_{\theta,n} < \sigma\delta_{f+n}$.

5.3. Tight Error Analysis

The analysis of scaling errors for the reciprocal square root is like that for the reciprocal. Again, we have that $b^* = av + \eta_1$, with $|\eta_1| < \delta_{f+n}$. And this time, we have

$$c_\theta^* = \frac{1}{\sqrt{b^*}} + \epsilon_{\theta,n},$$

where $|\epsilon_{\theta,n}| < \sigma\delta_{f+n}$ with $\sigma = 2.71$, according to Theorem 4. Finally, c^* is scaled back through the multiplication by $\sqrt{v}$ rounded (deterministically) to the original precision:

$$d_\theta^* = c_\theta^*\sqrt{v} + \eta_2,$$

where $|\eta_2| \leq \frac{1}{2}\delta_f$. The absolute error for $d = 1/\sqrt{a}$ then reads as

$$\begin{aligned}|d_\theta^* - d| &= \left|\left(\frac{1}{\sqrt{av+\eta_1}} + \epsilon_{\theta,n}\right)\sqrt{v} + \eta_2 - \frac{1}{\sqrt{a}}\right|\\ &= \left|\frac{1}{\sqrt{a}}\frac{1}{\sqrt{1+\frac{\eta_1}{av}}} + \epsilon_{\theta,n}\sqrt{v} + \eta_2 - \frac{1}{\sqrt{a}}\right|\\ &= \left|\frac{1}{\sqrt{a}}\left(1 - \frac{1}{2}\frac{\eta_1}{av} + \frac{3}{8}\left(\frac{\eta_1}{av}\right)^2 - \ldots\right) + \epsilon_{\theta,n}\sqrt{v} + \eta_2 - \frac{1}{\sqrt{a}}\right|\\ &\doteq \left|-\frac{\eta_1}{2\sqrt{a}b} + \epsilon_{\theta,n}\sqrt{v} + \eta_2\right|. \qquad (13)\end{aligned}$$

We are able to bound the overall error for $1/\sqrt{a}$ as follows, using Lemma A7 to bound the error for cases in which a is a specific power of 2.

Theorem 5. *If the Newton–Raphson method is used to compute* $1/\sqrt{a}$ *for* $a \in \mathbb{Q}_{2f,f}$*, employing the approach in Algorithm 5, with* $\frac{1}{2}f \leq n \leq f-2$*, then the absolute error (13) is bounded by* $(2^{(f-1)/2-n}\sigma + \frac{1}{2})\delta_f$.

Proof. Similar to Theorem 2, we can make a distinction between three cases: (i) $a < 2^{-2n}$, in which $\eta_1 = \eta_2 = 0$; (ii) $2^{-2n} \leq a < 2^{n+1}$, in which generally only $\eta_1 = 0$; and (iii) $a \geq 2^{n+1}$. However, since $\frac{1}{2}f \leq n$, which is equivalent to $2n \geq f$, there exists no $a \in \mathbb{Q}_{2f,f}$ such that $a < 2^{-2n}$. Therefore, we will only consider the cases (ii) and (iii).

In case (ii), the error simplifies to $|\epsilon_{\theta,n}\sqrt{v} + \eta_2|$. For the smallest value $a = \delta_f$, we find that the error is bounded by $(2^{f/2}\sigma\delta_{f+n} + \frac{1}{2}\delta_f)$ in the case that f is even, and by $(2^{(f-1)/2}\sigma\delta_{f+n} + \frac{1}{2}\delta_f)$ if f is odd. However, if f is even and $a = \delta_f$, we may replace $\sigma\delta_{f+n}$ by δ_{f+n}, according to Lemma A7. Therefore, a tighter bound for even f is found by considering the next value, $a = 2\delta_f$, for which the absolute error is bounded by $(2^{f/2-1}\sigma\delta_{f+n} + \frac{1}{2}\delta_f)$. However, the largest bound is found for an odd f and reads as $(2^{(f-1)/2-n}\sigma + \frac{1}{2})\delta_f$.

In case (iii), we need to take into account all error terms in (13). Without going into further detail, we state that the error is maximized by taking the lowest value of a in this range ($a = 2^{n+1}$), which also has the largest value for v, with $n+1$ even, because it maximizes the combined value of the first and second error terms. The absolute error then reads as $(2^{-(n+1)/2}(\frac{1}{2} + \sigma) + \frac{1}{2})\delta_f$.

It can be easily verified that with $n \leq f-2$, the bound in case (iii) is always below the (lowest) bound in case (ii). Therefore, for the given range of n, $(2^{(f-1)/2-n}\sigma + \frac{1}{2})\delta_f$ bounds the absolute error for all values of a. □

Remark 6. *The relative error is found by dividing the absolute error by* $d = 1/\sqrt{a}$*:*

$$\left|\frac{d_\theta^* - d}{d}\right| \doteq \left|-\frac{\eta_1}{2b} + \epsilon_{\theta,n}\sqrt{b} + \sqrt{a}\eta_2\right|.$$

Here, $\doteq$ *means equality up to higher-order terms. Note that this only applies to the term involving* η_1*, for which quadratic and higher-order terms are ignored. However, these terms do not have a significant effect in the cases with the largest absolute errors.*

The relative error is small for small values of a, while for larger values of a the error increases. If $a < 2^{-2n}$*, the error is bounded by* $2^{1/2-n}\sigma\delta_f$*, increasing to* $(2^{1/2-n}\sigma + 2^{(n-1)/2})\delta_f$ *for* $2^{-2n} \leq a < 2^{n+1}$*. For* $a \geq 2^{n+1}$*, the error is (approximately) bounded by* $(1 + 2^{1/2-n}\sigma + 2^{f/2-1})\delta_f$.

Corollary 3. *If the Newton–Raphson method is used to compute* $1/\sqrt{a}$ *for* $a \in \mathbb{Q}_{2f,f}$*, employing the approach in Algorithm 5, then computing with* $n = \lfloor (f+5)/2 \rfloor$ *additional bits guarantees that the absolute error (13) is strictly smaller than* δ_f.

Proof. According to Theorem 5, if $f/2 \leq n \leq f-2$, the absolute error is bounded by $(2^{(f-1)/2-n}\sigma + 1/2)\delta_f$. Thus, for the final absolute error to be smaller than δ_f, we need

$$(2^{(f-1)/2-n}\sigma + \tfrac{1}{2})\delta_f < \delta_f,$$

which gives that (approximately) $n > \frac{1}{2}f + 1.94$. From this, it follows that $n = \frac{1}{2}f + 2$ for even f, and $n = \frac{1}{2}f + \frac{5}{2}$ for odd f, which guarantees that the absolute error will be smaller than δ_f. Bearing in mind the range of n for which Theorem 5 is valid, this results holds for $f \geq 8$.

What remains to be shown is that: (i) smaller values for n will not suffice to guarantee an error smaller than δ_f, and (ii) the result also holds for $f < 8$. Both are achieved by exhaustively checking all rounding combinations for increasing values of f. For (ii), this shows that the results hold for $f \geq 4$ (all f that require at least one iteration). To prove (i),

we use $n = \frac{1}{2}f + 1$ for even f and $n = \frac{1}{2}f + \frac{3}{2}$ for odd f, until a final absolute error larger than δ_f is found. All $f \leq 300$ are checked, and several cases where the final error exceeds δ_f are identified, the first of which are $f = 20$ and $f = 223$. This proves that $n = \lfloor (f+5)/2 \rfloor$ is not only a sufficient condition, but (in general) also a necessary one. □

6. Square Root

Besides being a result in itself, the reciprocal square root can be used to compute the square root by multiplying it with the original input $[\![a]\!]$. In fact, this seems the most efficient way to achieve this, as it provides a means to compute the square root with multiplications and additions only, whereas applying the Newton–Raphson method to the square root directly would lead to an algorithm that requires the computation of a reciprocal (and hence a full Newton–Raphson computation) in every iteration.

6.1. Error for Square Root

Looking back at the computation of the reciprocal square root, after computing c_θ^*, we have several options. We could finish the computation of the reciprocal square root as we did before, multiplying c_θ^* by $\sqrt{v}$, and subsequently perform a rounding step. Because the multiplication by a will follow, it makes sense not to round to the original precision at this stage and keep the extra n bits to maintain a higher accuracy. Still, rounding to $f+n$ fractional bits induces an error that would be multiplied by a, which for large values of a would lead to a large error.

Instead, it is significantly better to multiply c_θ^* by a first, subsequently perform a rounding step, and only then multiply by $\sqrt{v}$. Even though the largest error is still attained for large values of a, at least we avoid multiplying the intermediate rounding error by this large a.

A third option, with an accuracy practically equal to the previous accuracy, is multiplying a and $\sqrt{v}$ separately (similar to multiplying g and v in Algorithm 3):

$$w = a\sqrt{v} + \eta_w.$$

Here, $|\eta_w| < \delta_{f+n}$. We then multiply c_θ^* with w and (deterministically) round the result to the original precision:

$$d_\theta^* = c_\theta^* w + \eta_2.$$

Again, $|\eta_2| < \frac{1}{2}\delta_f$. Subtracting the exact solution gives the absolute error:

$$\begin{aligned} |d_\theta^* - \sqrt{a}| &= \left| \left(\frac{1}{\sqrt{av + \eta_1}} + \epsilon_{\theta,n} \right) (a\sqrt{v} + \tilde{\eta}_1) + \eta_2 - \sqrt{a} \right| && (14) \\ &= \left| -\frac{\sqrt{a}\eta_1}{2av} \left(1 - \frac{3}{4}\frac{\eta_1}{av} + \frac{5}{8}\left(\frac{\eta_1}{av}\right)^2 - \dots \right) + \epsilon_{\theta,n} a\sqrt{v} + c^*\tilde{\eta}_1 + \eta_2 \right| \\ &< \left| -\frac{\sqrt{a}\eta_1}{2b} \left(1 + \frac{|\eta_1|}{b} \right) + \frac{\epsilon_{\theta,n} b}{\sqrt{v}} + c^*\tilde{\eta}_1 + \eta_2 \right|. && (15) \end{aligned}$$

The algorithm is summarized in Algorithm 6.

Algorithm 6 $\mathsf{Sqrt}([\![a]\!], n = 0)$ $\quad -2^{\ell-1} \leq a < 2^{\ell-1}$

Lines 1–10 of Algorithm 5
11: $[\![w]\!] \leftarrow \mathsf{Round}_{f-n}([\![a]\!][\![v^{\frac{1}{2}}]\!])$
12: $[\![d_\theta]\!] \leftarrow \mathsf{Round}_{f+2n}([\![c_\theta]\!][\![w]\!], \text{deterministic})$
13: **return** $[\![d_\theta]\!]$ ▷ $-2^{\ell-1} \leq d_\theta < 2^{\ell-1}$

Theorem 6. *If Algorithm 6 is used to compute $\sqrt{a}$ for $a \in \mathbb{Q}_{2f,f}$, with $n \geq \frac{1}{2}f$, then the absolute error (14) is bounded by $(2^{f/2-n}(\frac{1}{2}(1+\delta_{f+n}) + \sigma) + \frac{1}{2})\delta_f$ for even f and by $(2^{f/2-n}(\frac{1}{4}(1 + \frac{1}{2}\delta_{f+n}) + \sqrt{2}\sigma) + \frac{1}{2})\delta_f$ for odd f, where $\sigma = 2.71$.*

Proof. For $a < 2^n$, we have $\eta_1 = \widetilde{\eta}_1 = 0$, and the error simplifies to $|\epsilon_{\theta,n} b/\sqrt{v} + \eta_2|$. This is bounded by $(2^{(2-n)/2}\sigma + \frac{1}{2})\delta_f$ for even n, and by $(2^{(1-n)/2}\sigma + \frac{1}{2})\delta_f$ for odd n. However, the error increases for a larger a. For $a \geq 2^n$, the first term in (15) comes into play and increases with a. At the same time, a larger a generally leads to a smaller v, which increases the second term as well. Since $n \geq \frac{1}{2}f$, we still have that $\widetilde{\eta}_1 = 0$. Thus, the error bound simplifies to $E_{d^*_\theta}$, where

$$E_{d^*_\theta} = \left| -\frac{\sqrt{a}\eta_1}{2b}\left(1 + \frac{|\eta_1|}{b}\right) + \frac{\epsilon_{\theta,n} b}{\sqrt{v}} + \eta_2 \right|.$$

For $a \geq 2^{f-1}$ and even f, we have $v = 2^{-f}$, and we can write $a = 2^f b$, with $0.5 \leq b < 1$. Substituting this into the above equation gives

$$\begin{aligned} E_{d^*_\theta} &< \left(\frac{\sqrt{2^f b}}{2b}\left(1 + \frac{\delta_{f+n}}{b}\right) + \frac{\sigma b}{2^{-f/2}}\right)\delta_{f+n} + \tfrac{1}{2}\delta_f \\ &= 2^{f/2-n}\left(\frac{1}{2\sqrt{b}}\left(1 + \frac{\delta_{f+n}}{b}\right) + \sigma b\right)\delta_f + \tfrac{1}{2}\delta_f. \end{aligned}$$

The factor within the outer parentheses increases with b and can be bounded by choosing $b = 1$, which leads to the bound for even f. Notice that if $2^{f-2} \leq a < 2^{f-1}$, then indeed $1 \leq b < 2$. In this case, however, the $2^{f/2-n}$ factor would become $2^{f/2-n-1}$, making it twice as small, while the factor between parentheses would become less than twice as large. Thus, this would decrease the overall error.

For $a \geq 2^{f-1}$ and odd f, we have $v = 2^{-(f-1)}$, and we can write $a = 2^{f-1}b$, with $1 \leq b < 2$. Substituting this into the same equation gives

$$E_{d^*_\theta} < 2^{(f-1)/2-n}\left(\frac{1}{2\sqrt{b}}\left(1 + \frac{\delta_{f+n}}{b}\right) + \sigma b\right)\delta_f + \tfrac{1}{2}\delta_f.$$

Substituting $b = 2$ then yields the bound stated for odd f. Note that in this case choosing a smaller a would lead to a lower value for the factor in front of the parentheses, as well as a lower value for the factor in parentheses, and therefore it need not be considered. □

Corollary 4. *If Algorithm 6 is used to compute $\sqrt{a}$ for $a \in \mathbb{Q}_{2f,f}$, then computing with $n = \lfloor (f+7)/2 \rfloor$ additional bits guarantees that the absolute error (14) is strictly smaller than δ_f.*

Proof. Using the result of Theorem 6 for the case that f is even, the absolute error is certainly smaller than δ_f if the following holds:

$$2^{f/2-n}\left(\tfrac{1}{2}(1 + \delta_{f+n}) + \sigma\right) + \tfrac{1}{2} < 1.$$

To get rid of the δ_{f+n} term, we assign it a fairly large value, which, as we will see, does not matter much for the outcome. We choose $\delta_{f+n} = \frac{1}{16}$, which would be the correct value if $f + n = 4$. It then follows that (approximately) $n > \frac{1}{2}f + 2.69$, from which we conclude that $n = \frac{1}{2}f + 3$ is sufficient to guarantee an error smaller than δ_f. Based on simulation results, we suspect that $n = \frac{1}{2}f + 2$ would already be sufficient, as we were unable to find a case in which the error exceeded δ_f. However, since the largest error does not always occur for the same a value (as, for example, in the case of the reciprocal square root), simulations are very costly and could not be performed for large values of f. There are cases for which $n = \frac{1}{2}f + 1$ leads to errors larger than δ_f, so using this value for n clearly does not guarantee an error smaller than δ_f.

For odd values of f, an analogous derivation shows that using $n = \frac{1}{2}f + \frac{7}{2}$ is guaranteed to keep the final absolute error below δ_f. Also in this case, using one bit less seems already sufficient, since no counterexample could be found. Cases in which the error

exceeded δ_f were found for $n = \frac{1}{2}f + \frac{3}{2}$, which shows that this value for n is certainly not sufficient. □

6.2. Integer Square Root

A related primitive is the integer square root. For a given integer $[\![a]\!]$, this function computes integer $[\![q]\!]$ such that $q \leq \sqrt{a} < q + 1$; hence, $q = \lfloor \sqrt{a} \rfloor$. For this purpose, we may exploit the algorithm derived in the previous section.

Corollary 5. *Suppose Algorithm 6 is used to compute $\sqrt{a}$ for an integer $a \in \mathbb{Q}_{2f,f}$, with $f \geq 3$. If $\tilde{q}$ is rounded to an integer $\bar{q}$ deterministically, then no additional bits are required to guarantee that $\bar{q} \in \{q, q+1\}$.*

Proof. With the input a now an integer, we have that $\eta_1 = 0$. Also, $\eta_w = 0$. Thus, the error bound (15) simplifies to $|\epsilon_{\theta,n} b/\sqrt{v} + \eta_2|$. For even f, this is bounded by

$$\begin{aligned} |\epsilon_{\theta,n} b/\sqrt{v} + \eta_2| &< \frac{\sigma \delta_{f+n}}{\sqrt{v}} + \tfrac{1}{2}\delta_f \\ &= 2^{-f/2-n}\sigma + \tfrac{1}{2}\delta_f. \end{aligned}$$

Even without any extra bits ($n = 0$), this bound remains below 0.5 for $f \geq 3$, so that $q - 0.5 < \tilde{q} < q + 1.5$. Therefore, if $\tilde{q}$ is rounded deterministically to an integer $\bar{q}$, then $\bar{q} \in \{q, q+1\}$. Analogously, the same result can be shown to hold for odd f. □

Remark 7. *The result of Corollary 5 holds even if the rounding in line 12 of Algorithm 6 is performed probabilistically.*

After computing the square root of a and rounding to an integer, the correct solution q is recovered by a single secure comparison. The complete procedure is summarized in Algorithm 7.

Algorithm 7 $\mathsf{IntSqrt}([\![a]\!])$ $-2^{f-1} \leq a < 2^{f-1}$, with $a \in \mathbb{Z}$

Line 2 in Algorithm 6 simplifies to $[\![b]\!] \leftarrow 2^{-f+n}[\![a]\!][\![v]\!]$

1: $[\![\tilde{q}]\!] \leftarrow \mathsf{Sqrt}([\![a2^f]\!], n = 0)$ ▷ $-2^{\ell-1} \leq \tilde{q} < 2^{\ell-1}$

2: $[\![\bar{q}]\!] \leftarrow \mathsf{Round}_f([\![\tilde{q}]\!], \text{deterministic})$

3: $[\![q]\!] \leftarrow [\![\bar{q}]\!] - ([\![\bar{q}]\!]^2 > [\![a]\!])$

4: **return** $[\![q]\!]$ ▷ $-2^{f-1} \leq q < 2^{f-1}$

7. Conclusions

Basic secure fixed-point arithmetic allows for efficient $+, -, *, <$ operations and often easily extends to efficient dot products and matrix multiplications. The availability of efficient solutions for secure reciprocals and square roots opens up a much broader scope of applications, such as efficient solutions for secure Gaussian elimination, secure linear programming, and secure Cholesky decomposition with appropriately scaled input matrices.

As announced at the end of Section 2.1, our protocols achieve logarithmic round complexity: the round complexity is dominated by the $\theta = O(\log f)$ rounds for the for loops in Algorithms 2 and 5, as each iteration takes $O(1)$ rounds due to the use of probabilistic rounding. In concurrent work, we achieved similar results for the secure computation of sine and cosine in secure fixed-point arithmetic, relying on an iterative method very different from Newton-Raphson iteration, but also supporting any desired precision [20].

The use of secure fixed-point arithmetic is essential in many secure computation frameworks. As part of ongoing work, we are integrating all these solutions in the Python package MPyC [21], where the overall goal is to support all fixed-point arithmetic and

functions with arbitrary (parameterized) precision expressed as the number of fractional bits f. Our solutions for secure integer division (see Section 4) and secure integer square roots (see Section 6.2) therefore apply to secure integer arithmetic over arbitrarily large ranges. In fact, we use this form of secure integer division as a building block for the implementation of secure class groups in MPyC, see [22].

Author Contributions: Secure computation protocols, S.K. and B.S.; numerical analysis, S.K.; writing, S.K. and B.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Lemmas

This appendix collects all lemmas and proofs left out of the main text.

Appendix A.1. Lemmas for the Reciprocal

Lemma A1. *If the Newton–Raphson method is used to compute* $1/b$ *for some* $b \in [0.5, 1)$ *with exact arithmetic employing initial approximation* (2)*, and the number of iterations* $\theta \geq 1$ *is computed with* (6)*, then* $|\epsilon_{\theta-1}(b)| < b^{2^{\theta-1}-1}\sqrt{\delta_f}$.

Proof. The value for θ given by (6) is set such that $|\epsilon_\theta| < \delta_f$ holds for all b, and specifically for $b = 1$. From (5), it then follows that $\epsilon_0(1) < \delta_f^{2^{-\theta}}$. With initial approximation (2), we have $|\epsilon_0(b)| \leq \epsilon_0(1)$, and therefore $|\epsilon_0(b)| < \delta_f^{2^{-\theta}}$, for all b. Then, applying (5) to $\epsilon_0(b)$ with $i = \theta - 1$ gives the result. □

Lemma A2. *If the Newton–Raphson method is used to compute* $1/b$ *for some* $b \in [0.5, 1)$*, employing initial approximation* (2)*, then the rounding error in the first iteration is bounded by* $(c_0(b) + 1)\delta_f$*, while in any subsequent iteration it is bounded by* $(1/b + 1)\delta_f$.

Proof. Recall that for the reciprocal the iterative rule reads as

$$c_{i+1} = c_i(2 - c_i b),$$

in which the subtraction is carried out without rounding.

The first multiplication $c_i b = (c - \epsilon_i)b = 1 - \epsilon_i b$ yields after rounding:

$$\lfloor c_i b \rceil_\$ = 1 - \epsilon_i b + e_{i+1,1}, \tag{A1}$$

where $e_{i+1,1}$ is a probabilistic rounding term, with $|e_{i+1,1}| < \delta_f$ (see Section 2.3). The second multiplication gives

$$\begin{aligned} c_i(2 - \lfloor c_i b \rceil_\$) &= (c - \epsilon_i)(1 + \epsilon_i b - e_{i+1,1}) \\ &= c - c e_{i+1,1} - \epsilon_i^2 b + \epsilon_i e_{i+1,1} \\ &= c - \epsilon_i^2 b - c_i e_{i+1,1}, \end{aligned}$$

which is rounded to

$$c_{i+1} = \lfloor c_i(2 - \lfloor c_i b \rceil_\$) \rceil_\$ = c - \epsilon_i^2 b - c_i e_{i+1,1} + e_{i+1,2},$$

where $|e_{i+1,2}| < \delta_f$. Thus, instead of the "exact" result in (4), we now find

$$\epsilon_{i+1} = \epsilon_i^2 b + e_{i+1}, \tag{A2}$$

where e_{i+1} is the total rounding error resulting from iteration $i+1$:

$$e_{i+1} = c_i e_{i+1,1} - e_{i+1,2}.$$

Because $|e_{i+1,1}|$ and $|e_{i+1,2}|$ are strictly smaller than δ_f, it follows that $|e_1| < (c_0+1)\delta_f$. Moreover, using initial approximation (2), it holds that $0 < c_i \leq c$ for $i \geq 1$, and thus $|e_{i+1}| < (1/b+1)\delta_f$ for these values of i. □

Lemma A3. *If the Newton–Raphson method is used to compute* $1/a$ *for some* $a = \pm 2^\lambda \delta_f$, $\lambda \in \{0, 1, 2, \ldots, 2f-2\}$, *employing initial approximation* (2), *and the number of iterations* θ *is computed with* (6), *then* $|\epsilon_{\theta,n}| \leq 2\delta_{f+n}$. *In fact,* $\epsilon_{\theta,n} \in \{-\delta_{f+n}, 0, \delta_{f+n}, 2\delta_{f+n}\}$.

Proof. When $a = \pm 2^\lambda \delta_f$, with $\lambda \in \{0, 1, 2, \ldots, 2f-2\}$, a has exactly one nonzero bit. Consequently, a will be scaled to $b = 0.5$, for which $c = 2$ is an exact multiple of δ_{f+n}. Since the intermediate approximations c_i are also multiples of δ_{f+n}, the error terms $\epsilon_i = c - c_i$ will also be multiples of δ_{f+n}. As a result, the $\epsilon_i b$-term in (A1) is a multiple of $\frac{1}{2}\delta_{f+n}$, and it follows that the first rounding term in every iteration, $|e_{i+1,1}|$, is either zero or $\frac{1}{2}\delta_{f+n}$. If, for the moment, we omit the second rounding of the final iteration, then combining this knowledge with the analysis in the proof of Theorem 1 gives the maximal error:

$$E^{\diamond}_{3,15}(0.5)\,\delta_{f+n} = \left((0.5)^7 + 2(0.5)^4 T_3 \sqrt{\delta_{15}} + 0.5 T_3^2 \delta_{15} + 1\right)\delta_{f+n},$$

where $T_3(0.5) = (2\sqrt{2}+13)/4$. The diamond superscript indicates that another (probabilistic) rounding step is still to be performed. Computing the above value shows that it is only slightly above δ_{f+n}. Because the correct solution is an exact multiple of δ_{f+n}, the second rounding in the final iteration can only take the error as far as the next multiple of δ_{f+n}, which is $2\delta_{f+n}$.

By exhaustively checking all rounding combinations, this bound was found to also hold for the cases $f+n \leq 14$ with at least one iteration. Clearly, when $\theta = 0$, the error is already below δ_{f+n} to begin with and, since no iterations are performed, does not change.

So far, we have assumed that all errors point towards the positive direction. The situation is different if all rounding errors go in the negative direction. In this situation, the worst-case scenario would be that the iteration error after $\theta - 1$ iterations is zero, while all rounding errors in the final iteration are maximally negative. Again assuming that $|e_{i+1,1}|$ is either zero or $\frac{1}{2}\delta_{f+n}$, it then follows from (7) that

$$\min \epsilon^{\diamond}_{\theta}(0.5) = -\delta_{f+n}.$$

As before, the diamond superscript indicates that a final (probabilistic) rounding is still to be performed. Since the exact solution is still a multiple of δ_{f+n}, the second rounding in the final iteration cannot take the error any further than $-\delta_{f+n}$. This result is independent of the values of f and θ.

Combining $-\delta_{f+n} \leq \epsilon_\theta \leq 2\delta_{f+n}$ with the knowledge that the error is a multiple of δ_{f+n}, we find that $\epsilon_{\theta,n} \in \{-\delta_{f+n}, 0, \delta_{f+n}, 2\delta_{f+n}\}$. □

Lemma A4. *If the Newton–Raphson method is used to compute* $1/a$ *for* $a = \pm 3\delta_f$, *employing initial approximation* (2), *and the number of iterations* θ *is computed with* (6), *then* $|\epsilon_{\theta,n}| \leq \frac{7}{3}\delta_{f+n}$ *for* $f+n = 6$ *and* $|\epsilon_{\theta,n}| \leq \frac{5}{3}\delta_{f+n}$ *for all other values of* f *and* n.

Proof. When $a = \pm 3\delta_f$, a will be scaled to $b = 0.75$. With $b = 0.75$, $c = 4/3$, which is a multiple of $\frac{1}{3}\delta_{f+n}$. Since the intermediate approximations c_i are multiples of δ_{f+n}, the error terms $\epsilon_i = c - c_i$ will also be multiples of $\frac{1}{3}\delta_{f+n}$. As a result, the $\epsilon_i b$ term in (A1) is a multiple of $\frac{1}{4}\delta_{f+n}$, and it follows that the first rounding term in every iteration, $|e_{i+1,1}|$, can

be at most $\frac{3}{4}\delta_{f+n}$. If, for the moment, we omit the second rounding of the final iteration, then combining this knowledge with the analysis in the proof of Theorem 1 gives

$$E^{\diamond}_{3,15}(0.75)\,\delta_{f+n} = \Big((0.75)^7 + 2(0.75)^4 T_3\sqrt{\delta_{15}} + 0.75T_3^2\delta_{15} + 1\Big)\delta_{f+n},$$

where $T_3 = (c_0(0.75)+1) + (3-2)(0.75/0.75+1) = \sqrt{2}+3$, and the diamond superscript indicates that a final rounding is still to be performed. Computing the above value shows that it is slightly below $1.15\delta_{f+n}$. Because the correct solution c is a multiple of $\frac{1}{3}\delta_{f+n}$, the second rounding in the final iteration can only take the error as far as $\frac{5}{3}\delta_{f+n}$.

By exhaustively checking all rounding combinations, this bound is found to also hold for the cases $f+n \leq 14$ with at least one iteration, except for $f+n=6$, in which case $|\epsilon_{\theta,n}| = \frac{7}{3}\delta_{f+n}$ is the largest possible error. Also, here, when $\theta = 0$, the error is already below δ_{f+n} to begin with and does not change, since no iterations are performed. □

Appendix A.2. Lemmas for the Reciprocal Square Root

In the following lemma, we make use of the two points where $c_0(b) = c(b)$, which we hereby define as $b_1 \approx 0.58$ and $b_2 \approx 1.65$. Between these points, $\epsilon_0(b) < 0$, while outside the interval $\epsilon_0(b) > 0$.

Lemma A5. *If the Newton–Raphson method is used to compute $1/\sqrt{b}$ for some $b \in [0.5, 2)$ with exact arithmetic, employing initial approximation (9), and the number of iterations $\theta \geq 1$ is computed with (12), then $|\epsilon_{\theta-1}(b)| < \xi^{2^{\theta-2}}\sqrt{b/2}^{2^{\theta-1}-1}\sqrt{\delta_f/\tau}$, were $\tau = 3/\sqrt{2}$. The factor ξ may be taken as equal to 1.045 for $b_1 < b < b_2$ and unity elsewhere.*

Proof. The formula for θ in (12) is constructed in such a way that $\epsilon_\theta < \delta_f$ for $b = 2$. This is based on the assumption that $\epsilon_i = (\frac{3}{2}\sqrt{b})^{2^i-1}\epsilon_0^{2^i}$, which for $b = 2$ is a safe assumption. From this, it follows that $|\epsilon_0(2)| < \tau^{2^{-\theta}-1}\delta_f^{2-\theta}$. With the initial approximation (9), we have $|\epsilon_0(b)| \leq \epsilon_0(2)$, and therefore $|\epsilon_0(b)| < \tau^{2^{-\theta}-1}\delta_f^{2-\theta}$, for all b.

Next, consider the first iteration. Since there are inputs for which $\epsilon_0 < 0$, we cannot simply ignore the third-order term in (11). Instead, we have

$$\begin{aligned} |\epsilon_1| &= \left|\tfrac{3}{2}\sqrt{b}\epsilon_0^2 - \tfrac{1}{2}b\epsilon_0^3\right| \\ &\leq \tfrac{3}{2}\sqrt{b}\epsilon_0^2\big(1 + \tfrac{1}{3}\sqrt{b}|\epsilon_0|\big). \end{aligned}$$

The third-order term in the first line only has negative values for $b_1 < b < b_2$. For other values it is positive, meaning that it will only decrease the error, and can thus be safely ignored. Consequently, the largest value that the b term between parentheses in the second line might have is b_2. Combining this value with the largest initial error β (which actually do not coincide), we find that the term between parentheses is bounded by 1.045, and we obtain

$$|\epsilon_1| \leq \tfrac{3}{2}\xi\sqrt{b}\big(\tau^{2^{-\theta}-1}\delta_f^{2-\theta}\big)^2,$$

where $\xi = 1.045$ for $b_1 < b < b_2$ and unity elsewhere.

We know that after the first iteration, $\epsilon_i > 0$. Therefore, the third-order term in (11) will be larger than zero, and we have $\epsilon_{i+1} < \frac{3}{2}\sqrt{b}\epsilon_i^2$. Applying this for the remaining $\theta - 2$ iterations gives

$$\begin{aligned} |\epsilon_{\theta-1}| &< (\tfrac{3}{2}\sqrt{b})^{2^{\theta-2}-1}\epsilon_1^{2^{\theta-2}} \\ &\leq (\tfrac{3}{2}\sqrt{b})^{2^{\theta-2}-1}\Big(\tfrac{3}{2}\xi\sqrt{b}\big(\tau^{2^{-\theta}-1}\delta_f^{2-\theta}\big)^2\Big)^{2^{\theta-2}} \\ &= \xi^{2^{\theta-2}}\sqrt{b/2}^{2^{\theta-1}-1}\sqrt{\delta_f/\tau}, \end{aligned}$$

which proves the statement. □

Next, we consider the case in which the result of every multiplication is rounded. For the reciprocal square root, there are three multiplications per iteration.

Lemma A6. *If the Newton–Raphson method is used to compute* $1/\sqrt{b}$ *for some* $b \in [0.5, 2)$*, employing initial approximation* (9)*, then the rounding error in the first iteration is bounded by* $(c_0^2/2 + c_0/2 + 1)\delta_f$*, while in any subsequent iteration it is bounded by* $(1/(2b) + 1/(2\sqrt{b}) + 1)\delta_f$*.*

Proof. Recall that for the reciprocal square root, the iterative rule reads as

$$c_{i+1} = \tfrac{1}{2} c_i \Big(3 - c_i^2 b\Big).$$

The first multiplication gives

$$\begin{aligned} c_i b &= (c - \epsilon_i) b \\ &= \sqrt{b} - \epsilon_i b, \end{aligned}$$

which is subsequently rounded to

$$\lfloor c_i b \rceil_\$ = \sqrt{b} - \epsilon_i b + e_{i+1,1}, \tag{A3}$$

with probabilistic rounding error $|e_{i+1,1}| < \delta_f$. Next, we perform the second multiplication:

$$\begin{aligned} c_i \lfloor c_i b \rceil_\$ &= (c - \epsilon_i)\Big(\sqrt{b} - \epsilon_i b + e_{i+1,1}\Big) \\ &= 1 - 2\sqrt{b}\epsilon_i + \epsilon_i^2 b + c_i e_{i+1,1}, \end{aligned}$$

which is then rounded to

$$\big\lfloor c_i \lfloor c_i b \rceil_\$ \big\rceil_\$ = 1 - 2\sqrt{b}\epsilon_i + \epsilon_i^2 b + c_i e_{i+1,1} + e_{i+1,2},$$

where $|e_{i+1,2}| < \delta_f$. The subtraction that follows is without rounding. The third and final multiplication gives

$$\begin{aligned} \tfrac{1}{2} c_i \Big(3 - \big\lfloor c_i \lfloor c_i b \rceil_\$ \big\rceil_\$\Big) &= \tfrac{1}{2}(c - \epsilon_i)\Big(2 + 2\sqrt{b}\epsilon_i - \epsilon_i^2 b - c_i e_{i+1,1} - e_{i+1,2}\Big) \\ &= c - \tfrac{3}{2}\sqrt{b}\epsilon_i^2 + \tfrac{1}{2} b \epsilon_i^3 - \tfrac{1}{2} c_i^2 e_{i+1,1} - \tfrac{1}{2} c_i e_{i+1,2}, \end{aligned}$$

which is rounded to

$$c_{i+1} = \Big\lfloor \tfrac{1}{2} c_i \Big(3 - \big\lfloor c_i \lfloor c_i b \rceil_\$ \big\rceil_\$\Big) \Big\rceil_\$ = c - \tfrac{3}{2}\sqrt{b}\epsilon_i^2 + \tfrac{1}{2} b \epsilon_i^3 - \tfrac{1}{2} c_i^2 e_{i+1,1} - \tfrac{1}{2} c_i e_{i+1,2} + e_{i+1,3}.$$

Again, $|e_{i+1,3}| < \delta_f$. Thus, instead of the "exact" result in (11), we now find

$$\epsilon_{i+1} = \tfrac{3}{2}\sqrt{b}\epsilon_i^2 - \tfrac{1}{2} b \epsilon_i^3 + e_{i+1}, \tag{A4}$$

with

$$e_{i+1} = \tfrac{1}{2} c_i^2 e_{i+1,1} + \tfrac{1}{2} c_i e_{i+1,2} - e_{i+1,3}.$$

Because all rounding terms are strictly smaller than δ_f (see Section 2.3), it directly follows that $e_1 < (c_0^2/2 + c_0/2 + 1)\delta_f$. Moreover, with the approximation (9), we have $0 < c_i \leq c$, and therefore $|e_{i+1}| < (1/(2b) + 1/(2\sqrt{b}) + 1)\delta_f$, for $i \geq 1$. □

Lemma A7. *If the Newton–Raphson method is used to compute* $1/\sqrt{a}$ *for some* $a = \pm 2^\lambda \delta_f$*, with* $\lambda \in \{0, 2, 4, \ldots, 2f - 2\}$ *and even* f*, or with* $\lambda \in \{1, 3, 5, \ldots, 2f - 3\}$ *and odd* f*, employing*

initial approximation (9), *and the number of iterations* θ *is computed with* (12), *then* $|\epsilon_{\theta,n}| \leq \delta_{f+n}$. *In particular,* $\epsilon_{\theta,n} \in \{-\delta_{f+n}, 0, \delta_{f+n}\}$.

Proof. When $a = \pm 2^{\lambda}\delta_f$, with $\lambda \in \{0, 2, \ldots, 2f-2\}$ and even f, or with $\lambda \in \{1, 3, \ldots, 2f-3\}$ and odd f, a will be scaled to $b = 1$. Then, $c = 1$, which is an exact multiple of δ_{f+n}. Since the intermediate approximations c_i are also multiples of δ_{f+n}, the error terms $\epsilon_i = c - c_i$ will also be multiples of δ_{f+n}. As a result, the $\epsilon_i b$ term in (A3) is a multiple of δ_{f+n}, and it follows that the first rounding term in every iteration, $|e_{i+1,1}|$, is zero. If, for the moment, we omit the third rounding term of the final iteration, then combining this knowledge with the analysis in the proof of Theorem 4 gives the maximal error:

$$E^{\diamond}_{4,19}(1)\,\delta_{f+n} = \left(\sqrt{\xi}\sqrt{\xi/2}^{15} + 2\sqrt{\xi/2}^{8} T_4 \sqrt{\tau\delta_{19}} + \tfrac{3}{2}T_4^2\delta_{19} + \frac{1}{2}\right)\delta_{f+n},$$

where $\xi = 1.045$, $T_4 = (c_0(1)/2 + 1) + (4-2)(1/2+1) = \frac{1}{8}(35 + \sqrt{2})$, and the diamond superscript indicates that another rounding step is still to be performed. A simple numerical evaluation of the above expression shows that its value is slightly below $0.51\delta_{f+n}$. Because the correct solution c is an exact multiple of δ_{f+n}, the second rounding in the final iteration can only take the error as far as the next multiple of δ_{f+n}, which is δ_{f+n}. By exhaustively checking all rounding combinations, this bound is found to also hold for the cases $f + n \leq 18$ with at least one iteration. Clearly, when $\theta = 0$, the error is already below δ_{f+n} to begin with and, since no iterations are performed, does not change.

Combining $-\delta_{f+n} \leq \epsilon_\theta \leq \delta_{f+n}$ with the knowledge that the error is a multiple of δ_{f+n}, we find that $\epsilon_{\theta,n} \in \{-\delta_{f+n}, 0, \delta_{f+n}\}$. Numerical experiments further suggest that actually $\epsilon_{\theta,n} \in \{0, \delta_{f+n}\}$, but we will not prove this here. □

References

1. Algesheimer, J.; Camenisch, J.; Shoup, V. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *Advances in Cryptology—CRYPTO 2002*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2442, pp. 417–432.
2. Catrina, O.; de Hoogh, S. Secure multiparty linear programming using fixed-point arithmetic. In *Computer Security—ESORICS 2010*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6345, pp. 134–150.
3. Catrina, O.; Saxena, A. Secure computation with fixed-point numbers. In *Financial Cryptography and Data Security—FC 2010*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6052, pp. 35–50.
4. Liedel, M. Secure distributed computation of the square root and applications. In *Information Security Practice and Experience—ISPEC 2012*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7232, pp. 277–288.
5. Aly, A.; Smart, N.P. Benchmarking privacy preserved scientific operations. In *Applied Cryptography and Network Security—ACNS 2019*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2019; Volume 11464, pp. 509–529.
6. Knuth, D.E. *The Art of Computer Programming (Vol. 2: Seminumerical Algorithms)*, 3rd ed.; Addison Wesley: Reading, MA, USA, 1997.
7. Wilkinson, J.H. *Rounding Errors in Algebraic Processes*; Prentice Hall: Englewood Cliffs, NJ, USA, 1963.
8. Wilkinson, J.H. The algebraic eigenvalue problem. In *Monographs on Numerical Analysis*; Clarendon Press: Oxford, UK, 1965.
9. Aly, A.; Nawaz, K.; Salazar, E.; Sucasas, V. Through the looking-glass: Benchmarking secure multi-party computation comparisons for ReLU's. In *Cryptology and Network Security—CANS 2022*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2022; Volume 13641, pp. 44–67.
10. Damgård, I.; Fitzi, M.; Kiltz, E.; Nielsen, J.B.; Toft, T. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference—TCC 2006*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2006; Volume 3876, pp. 285–304.
11. Damgård, I.; Nielsen, J.B. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology—CRYPTO 2003*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2729, pp. 247–264.
12. Croci, M.; Giles, M.B. Effects of round-to-nearest and stochastic rounding in the numerical solution of the heat equation in low precision. *IMA J. Numer. Anal.* **2022**, *43*, 1358–1390. [CrossRef]
13. Na, T.; Ko, J.H.; Kung, J.; Mukhopadhyay, S. On-chip training of recurrent neural networks with limited numerical precision. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; pp. 3716–3723.

14. Paxton, E.A.; Chantry, M.; Klöwer, M.; Saffin, L.; Palmer, T. Climate modeling in low precision: Effects of both deterministic and stochastic rounding. *J. Clim.* **2022**, *35*, 1215–1229. [CrossRef]
15. Wang, N.; Choi, J.; Brand, D.; Chen, C.; Gopalakrishnan, K. Training deep neural networks with 8-bit floating point numbers. In Proceedings of the 32nd International Conference on Neural Information Processing Systems—NIPS 2018, Santa Barbara, CA, USA, 18–22 August 2002; Curran Associates, Inc.: Red Hook, NY, USA, 2018; pp. 7686–7695.
16. Croci, M.; Fasi, M.; Higham, N.J.; Mary, T.; Mikaitis, M. Stochastic rounding: Implementation, error analysis and applications. *R. Soc. Open Sci.* **2022**, *9*, 211631. [CrossRef]
17. Ryaben'kii, V.S.; Tsynkov, S.V. *A Theoretical Introduction to Numerical Analysis*; Chapman and Hall/CRC: New York, NY, USA, 2006.
18. Yamamoto, T. Historical developments in convergence analysis for Newton's and Newton-like methods. *J. Comput. Appl. Math.* **2000**, *124*, 1–23. [CrossRef]
19. Ercegovac, M.; Lang, T. *Digital Arithmetic*; Morgan Kaufmann: San Francisco, CA, USA, 2004.
20. Korzilius, S.; Schoenmakers, B. New approach for sine and cosine in secure fixed-point arithmetic. In *Cyber Security, Cryptology, and Machine Learning—CSCML 2023*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2023; Volume 13914, pp. 307–319.
21. Schoenmakers, B. MPyC Package for Secure Multiparty Computation in Python. GitHub. 2018. Available online: github.com/lschoe/mpyc (accessed on 7 September 2023).
22. Schoenmakers, B.; Segers, T. Efficient Extended GCD and Class Groups from Secure Integer Arithmetic. In *Cyber Security, Cryptology, and Machine Learning—CSCML 2023*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2023; Volume 13914, pp. 32–48.