SQUARE ROOT

1. INTRODUCTION

Let $\tilde{x}$ be a nonnegative integer, and consider

$$\tilde{y} = \sqrt{\tilde{x}}.$$

Then $\tilde{y}$ is irrational unless $\tilde{x}$ is a perfect square, and so in general cannot be represented in floating-point form, let alone integer form. We cannot verify the equality $\tilde{y} = \sqrt{\tilde{x}}$ in an arithmetic circuit. Instead, we verify that

$$y = \lfloor \sqrt{\tilde{x}} \rfloor,$$

which reduces to a number of range checks. Specifically, this relation holds if and only if

$$y \geqslant 0 \quad \text{and} \quad y^2 \leqslant \tilde{x} < (y+1)^2.$$

The error introduced by replacing $\sqrt{\tilde{x}}$ with its integer part may be too large for some applications. To improve accuracy, we scale $\tilde{x}$ by a factor of α^2 and verify that $y = \lfloor \sqrt{x} \rfloor$, where $x = \alpha^2 \tilde{x}$. We then interpret y as an approximation to $\alpha\tilde{y} = \alpha\sqrt{\tilde{x}}$.

Finally, if $\tilde{x}$ is not an integer, we define $x = \lfloor \alpha^2 \tilde{x} \rfloor$. This substitution does not affect the value of $\lfloor \sqrt{x} \rfloor$, but it allows us to replace a real-valued function of real parameters with an integer-valued function of integer parameters—enabling verification within an arithmetic circuit.

2. THE PROCESS

Let $\tilde{x}$ be a nonnegative real number, and suppose

$$\tilde{y} = \sqrt{\tilde{x}}. \tag{2.1}$$

In general, $\tilde{y}$ may be irrational, and therefore may not be exactly represented in floating-point or integer form. A common workaround is to replace $\tilde{y}$ with its integer floor $\lfloor \sqrt{\tilde{x}} \rfloor$. In fact, by Proposition 3.1, $\lfloor \sqrt{\tilde{x}} \rfloor = \lfloor \sqrt{x} \rfloor$, where $x = \lfloor \tilde{x} \rfloor$, and so (2.1) may be replaced by an integer-valued function that takes integer inputs.

For $\tilde{x} > 0$, this gives an approximation with relative error

$$\frac{\sqrt{\tilde{x}} - \lfloor \sqrt{\tilde{x}} \rfloor}{\lfloor \sqrt{\tilde{x}} \rfloor} = \frac{\{\sqrt{\tilde{x}}\}}{\lfloor \sqrt{\tilde{x}} \rfloor},$$

where $\{\sqrt{\tilde{x}}\} = \sqrt{\tilde{x}} - \lfloor \sqrt{\tilde{x}} \rfloor \in [0,1)$ denotes the fractional part of $\sqrt{\tilde{x}}$. When $\tilde{x} > 0$, this relative error is less than $2/\sqrt{\tilde{x}}$, because for real numbers $z \geqslant 1$ in general,

$$\frac{\{z\}}{\lfloor z \rfloor} = \frac{\{z\}}{z} \cdot \frac{z}{\lfloor z \rfloor} < \frac{1}{z} \cdot \frac{\lfloor z \rfloor + 1}{\lfloor z \rfloor} \leqslant \frac{2}{z}.$$

To improve the quality of this approximation, we introduce a positive integer scaling factor α and define

$$x = \lfloor \alpha^2 \tilde{x} \rfloor, \quad y = \lfloor \sqrt{x} \rfloor.$$

Since $\lfloor \sqrt{x} \rfloor = \lfloor \sqrt{\alpha^2 \tilde{x}} \rfloor$ by Proposition 3.1, we have

$$\sqrt{\tilde{x}} = \frac{\sqrt{\alpha^2 \tilde{x}}}{\alpha} = \frac{\lfloor \sqrt{\alpha^2 \tilde{x}} \rfloor}{\alpha} \left(1 + \frac{\{\sqrt{\alpha^2 \tilde{x}}\}}{\lfloor \sqrt{\alpha^2 \tilde{x}} \rfloor}\right) = \frac{\lfloor \sqrt{x} \rfloor}{\alpha} \left(1 + \frac{\theta}{\alpha\sqrt{\tilde{x}}}\right) \quad \text{for some} \quad \theta \in [0,2), \tag{2.2}$$

provided $\tilde{x} > 0$.

This shows that the rational approximation of $\sqrt{\tilde{x}}$ by $\lfloor\sqrt{x}\rfloor/\alpha$ introduces a relative error of order $1/\left(\alpha\sqrt{\tilde{x}}\right)$, improving on the integral approximation $\lfloor\sqrt{\lfloor\tilde{x}\rfloor}\rfloor$ by a factor of α. Even so, (2.2) is not a good approximation unless $\alpha^2\tilde{x} \gg 1$. Therefore, if possible, we choose α such that this holds.

Although we introduced the rational approximation $\lfloor\sqrt{x}\rfloor/\alpha$ to motivate the scaling step, we do not verify this relation directly in the circuit. Instead, we operate entirely on integers, verifying the simpler statement

$$y = \lfloor\sqrt{x}\rfloor,$$

where $x = \lfloor\alpha^2\tilde{x}\rfloor$. In doing so, we effectively "absorb" the scaling into the witness values. It is only through the off-circuit interpretation—viewing x as a scaled version of $\tilde{x}$ and y as an approximation to $\alpha\sqrt{\tilde{x}}$—that we recover the intended rational semantics. The arithmetic circuit itself remains oblivious to this interpretation, enforcing only integer relations.

2.1. Steps in the process. Each step in the process has a corresponding explanatory note in Subsection 2.2 that provides additional context and details.

(1) The circuit operates over the finite field $\mathbb{Z}/p\mathbb{Z}$, where p is a prime.

(2) Choose an integer base $b \geqslant 2$, a nonnegative integer κ, and an integer h, such that

$$b^{2\kappa} \leqslant h \leqslant \frac{p+1}{2}. \tag{2.3}$$

(3) Let x and y be integers lying in the interval of length p with left endpoint $h-p$, which defines a complete set of residues mod p:

$$h - p \leqslant x, y < h. \tag{2.4}$$

(4) The goal of the circuit is to verify that

$$x \geqslant 0 \quad \text{and} \quad y = \lfloor\sqrt{x}\rfloor. \tag{2.5}$$

(5) As most frameworks work exclusively with least residue representations, we define $\bar{z}$ as the least residue of z modulo p:

$$\bar{x} \equiv x \bmod p, \quad \bar{y} \equiv y \bmod p, \quad 0 \leqslant \bar{x}, \bar{y} < p.$$

The prover supplies $\bar{x}$ and $\bar{y}$ to the circuit as (private or public) witnesses.

(6) Perform two range checks to ensure that

$$0 \leqslant \bar{x} < b^\kappa \quad \text{and} \quad 0 \leqslant \bar{y} < b^\kappa. \tag{2.6}$$

These range checks are performed in the circuit using witness-supplied base-b digits for $\bar{x}$ and $\bar{y}$ (see Comment (6)).

(7) Under the assumptions (2.3) and (2.4), the bounds (2.6) imply [Proposition 3.2(a)] that

$$x = \bar{x} \quad \text{and} \quad y = \bar{y}, \tag{2.7}$$

as well as that

$$h - p \leqslant x - y^2, y^2 + 2y - x < h. \tag{2.8}$$

(8) The circuit computes the least residues $\overline{x-y^2}$ and $\overline{y^2+2y-x}$ of $x-y^2$ and y^2+2y-x, modulo p, respectively:

$$\overline{x-y^2} \equiv \bar{x} - \bar{y}^2 \bmod p \quad \text{and} \quad \overline{y^2+2y-x} \equiv \bar{y}^2 + 2\bar{y} - \bar{x} \bmod p. \tag{2.9}$$

These field operations (squaring, addition, subtraction) are computed directly inside the circuit, using the least residue representations $\bar{x}$ and $\bar{y}$ as inputs.

(9) Perform two further range checks to ensure that

$$0 \leqslant \overline{x-y^2} < b^\kappa \quad \text{and} \quad 0 \leqslant \overline{y^2+2y-x} < b^\kappa. \tag{2.10}$$

The prover supplies base-b decompositions of these two expressions as additional witness values, and the circuit performs range checks as in Step (6).

(10) Under assumption (2.3), the bounds (2.8) and (2.10) imply [Proposition 3.2(b)] that

$$x - y^2 = \overline{x-y^2} \quad \text{and} \quad y^2 + 2y - x = \overline{y^2+2y-x}. \tag{2.11}$$

(11) Combining (2.6), (2.7), (2.10), and (2.11), we see that

$$0 \leqslant x, y, x-y^2, y^2+2y-x < b^\kappa \tag{2.12}$$

These bounds imply that

$$y \geqslant 0 \quad \text{and} \quad y^2 \leqslant x < (y+1)^2, \tag{2.13}$$

which is equivalent to (2.5). Conversely, if (2.13) holds and $(y+1)^2 \leqslant b^\kappa$, then [Proposition 3.2(c)] (2.6) and (2.11) hold.

2.2. Commentary on each step.

(1) Typically, p is an n-bit prime satisfying

$$2^{n-1} \leqslant p < 2^n,$$

with $n \approx 256$. In practice, we use the prime field associated with the scalar field of the BN254 elliptic curve, a 254-bit prime that offers a good balance between security and efficiency. This field is widely supported in cryptographic applications and zk-SNARK frameworks due to the curve's pairing-friendly properties and efficient arithmetic over $\mathbb{Z}/p\mathbb{Z}$.

Polyhedra Network's EXPANDER prover and EXPANDERCOMPILERCOLLECTION (ECC) [3, 4, 5] support both the BN254 prime and the 31-bit Mersenne prime $2^{31}-1$. Although the latter is far too small for cryptographic security on its own, ECC uses internal field extensions to enable meaningful production use over this field, trading off cryptographic hardness for performance and recursion flexibility in certain use cases.

(2) Parameters b, κ, and h are circuit constants. We typically choose $b = 2$ (unless we use a lookup table in our range check steps ((6), (9)). The exponent κ should be chosen as small as possible to minimize circuit size, but note that we require $(y+1)^2 \leqslant b^\kappa$ for all valid y (see Step (11)), otherwise correct results may fail verification.

The mathematics is valid if we replace b^κ by any positive integer c satisfying $c^2 \leqslant h$ and $(y+1)^2 \leqslant c$. However, our range check method relies on expressing integers in base b and verifying their validity via their κ least significant base-b digits (See Comment (6)). This approach only works cleanly when c is a perfect power of b, ensuring that the representation fully determines membership in $[0, b^\kappa)$.

See Comment (3) regarding h. In this scenario, we typically choose $h = (p+1)/2$. Note that $h = (p+1)/2$ if and only if $h - p = -(p-1)/2$, in which case (2.4) becomes

$$-\frac{p-1}{2} \leqslant x, y \leqslant \frac{p-1}{2}.$$

If off-circuit reasoning does preclude the possibility that x and y range over negative and positive values, or that they tend to be closer to zero if of a certain sign, then this is a reasonable assumption.

In certain circumstances, we may be able to assume that $0 \leqslant x, y < p$ at the outset. However, note that in Step (7) we also need to assume that $x - y^2$ and $y^2 + 2y - x$, which may well be negative, lie in a certain interval of length p. For simplicity, we choose a value of h that leads to the desired conclusions in both Step (7) and Step (10).

(3) The assumption that $x, y \in [h-p, h)$ is a hypothesis of Proposition 3.2, which is invoked in Steps (7), (10), and (11). Arithmetic circuits over a finite field of order p can only impose constraints on the least residues $\bar{x}, \bar{y}$ of x, y modulo p. Without an underlying assumption about the range for x, y, we cannot translate constraints on $\bar{x}, \bar{y}$ into results for x, y.

Since an arithmetic circuit over a field of order p cannot distinguish between integers that differ by a multiple of p, the assumption that $x, y \in [h-p, h)$ cannot be enforced within the circuit and must instead be justified through off-circuit reasoning.

(4) We have $y = \lfloor \sqrt{x} \rfloor$ if and only if

$$y \leqslant \sqrt{x} < y + 1,$$

which, in turn, is equivalent to

$$y \geqslant 0 \quad \text{and} \quad y^2 \leqslant x < (y+1)^2.$$

Thus, verifying an integer square root reduces to a number of range checks.

Of course, $\lfloor \sqrt{x} \rfloor$ is an integer approximation to $\sqrt{x}$. If x is not a perfect square, $\sqrt{x}$ is irrational, and we cannot represent it in floating point, let alone within an arithmetic circuit. For $x \geqslant 1$, we have

$$\sqrt{x} = \lfloor \sqrt{x} \rfloor \left(1 + \frac{\{\sqrt{x}\}}{\sqrt{x}} \right).$$

Here, $\{\sqrt{x}\}$ denotes the fractional part of $\sqrt{x}$, which lies in $[0, 1)$. Thus, if x is a large integer, the error in the approximation of $\sqrt{x}$ by $\lfloor \sqrt{x} \rfloor$ is relatively small.

(5) These values may be private (e.g., for hidden inputs) or public (if the square root relation is to be exposed). In either case, the prover supplies $\bar{x}$ and $\bar{y}$, and the circuit treats them as witness elements. Since the relation $\bar{y} = \lfloor \sqrt{\bar{x}} \rfloor$ cannot be expressed using field arithmetic alone, the value of y must be computed off-circuit using a *hint* during witness generation. Hints in the EXPANDERCOMPILERCOLLECTION framework allow the prover to supply witness values derived from non-algebraic computations such as square roots, rounding, or conditionals, which are then checked inside the circuit via auxiliary constraints.

(6) See [2] for full details and (pseudo)code related to our range check process. We provide an outline of the process here: apply with $a \in \{x, y\}$.

(6a) Assume that $h - p \leqslant a < h$, where $b^\kappa \leqslant h \leqslant p$, so that $a = \bar{a}$ if and only if $0 \leqslant \bar{a} < h$ (see (3.10)), and $a \in [0, b^\kappa)$ if and only if $\bar{a} \in [0, b^\kappa)$.

(6b) The prover computes the κ least significant base-b digits of $\bar{a}$, and supplies the ordered tuple $(d_{\kappa-1}, \ldots, d_0)$ to the circuit as a (public or private) witness:

$$\bar{a} = b^\kappa q_\kappa + b^{\kappa-1} d_{\kappa-1} + \cdots + b^0 d_0, \quad q_\kappa \in \mathbb{Z}, \quad d_j \in \{0, 1, \ldots, b-1\}, \quad 0 \leqslant j < \kappa. \tag{2.14}$$

(6c) Impose constraints in the arithmetic circuit:

- For $0 \leqslant j < \kappa$, ensure $d_j \equiv c_j \bmod p$ with $c_j \in \{0, 1, \ldots, b-1\}$, either by enforcing

$$d_j(d_j - 1) \cdots (d_j - (b-1)) \equiv 0 \bmod p, \tag{2.15}$$

  or by enforcing membership in a lookup table (LUT) containing the valid digits $\{0, 1, \ldots, b-1\}$.
- Require

$$\bar{a} \equiv b^{\kappa-1} d_{\kappa-1} + \cdots + b^0 d_0 \bmod p. \tag{2.16}$$

(6d) Since $0 \leqslant \bar{a} < p$ and $0 \leqslant b^{\kappa-1} d_{\kappa-1} + \cdots + b^0 d_0 \leqslant b^\kappa - 1 < p$, these constraints guarantee

$$\bar{a} = b^{\kappa-1} d_{\kappa-1} + \cdots + b^0 d_0, \tag{2.17}$$

provided each d_j is taken as a least residue, and hence that $\bar{a}$ lies in the interval $[0, b^\kappa)$.

(7) The assumption (2.3) plays a crucial role here—see the proof of Proposition 3.2(a).

(8) These expressions are computed in the circuit using basic field operations. Since all values are represented as least residues, no special reduction is necessary—subtraction and multiplication behave as expected modulo

(9) See Comment (6): repeat the range check steps for $a \in \{x - y^2, y^2 + 2y - x\}$.

(10) For this particular result, the assumption (2.3) can be weakened to $b^\kappa \leqslant h \leqslant p$.

(11) See Comment (4). Of course, we do not need the upper bound in (2.12) in order to deduce (2.13) from here. The upper bound is a byproduct of (2.10) and (2.11).

It is also important to verify the converse result, as we do not want any correct results to fail verification. It suffices to assume that $(y+1)^2 \leqslant b^\kappa$ (see the proof of Proposition 3.2(c) for details). This must be taken into consideration in Step (2), when parameters b and κ are chosen. As these are circuit constants, we do not need to verify that $(y+1)^2 \leqslant b^\kappa$ for all valid y.

3. THE MATHS

Proposition 3.1. *For all real numbers $z \geqslant 0$, we have*

$$\lfloor \sqrt{z} \rfloor = \lfloor \sqrt{\lfloor z \rfloor} \rfloor. \tag{3.1}$$

Proof. Since $\lfloor z \rfloor \leqslant z$, we have $\sqrt{\lfloor z \rfloor} \leqslant \sqrt{z}$, and consequently,

$$\lfloor \sqrt{\lfloor z \rfloor} \rfloor \leqslant \lfloor \sqrt{z} \rfloor. \tag{3.2}$$

For the reverse inequality, note that since

$$\sqrt{\lfloor z \rfloor} < \lfloor \sqrt{\lfloor z \rfloor} \rfloor + 1,$$

we have

$$\lfloor z \rfloor < \left(\lfloor \sqrt{\lfloor z \rfloor} \rfloor + 1 \right)^2.$$

Since both sides of this inequality are integers, this implies

$$\lfloor z \rfloor + 1 \leqslant \left(\lfloor \sqrt{\lfloor z \rfloor} \rfloor + 1 \right)^2$$

Upon taking square roots, this implies

$$\sqrt{\lfloor z \rfloor + 1} \leqslant \lfloor \sqrt{\lfloor z \rfloor} \rfloor + 1.$$

Now,

$$\lfloor\sqrt{z}\rfloor \leqslant \sqrt{z} < \sqrt{\lfloor z\rfloor + 1},$$

because $z < \lfloor z\rfloor + 1$. Combining these last two sets of inequalities, we see that

$$\lfloor\sqrt{z}\rfloor < \lfloor\sqrt{\lfloor z\rfloor}\rfloor + 1.$$

Finally, since both sides of this inequality are integers, this implies that

$$\lfloor\sqrt{z}\rfloor \leqslant \lfloor\sqrt{\lfloor z\rfloor}\rfloor. \tag{3.3}$$

Combining (3.2) with (3.3) yields (3.1). □

Proposition 3.2. *Let p be an odd integer (not necessarily prime), and let $b \geqslant 2$, $\kappa \geqslant 0$, and h be integers such that*

$$b^{2\kappa} \leqslant h \leqslant \frac{p+1}{2}. \tag{3.4}$$

Assume that integers x and y satisfy

$$h - p \leqslant x, y < h. \tag{3.5}$$

Finally, let $\bar{x}$ and $\bar{y}$ denote the least residues of x and y modulo p:

$$\bar{x} \equiv x \bmod p, \quad \bar{y} \equiv y \bmod p, \quad 0 \leqslant \bar{x}, \bar{y} < p.$$

(a) If

$$0 \leqslant \bar{x}, \bar{y} < b^{\kappa}, \tag{3.6}$$

then $x = \bar{x}$, $y = \bar{y}$, and

$$h - p \leqslant x - y^2, y^2 + 2y - x < h. \tag{3.7}$$

(b) Let $\overline{x-y^2}$ and $\overline{y^2+2y-x}$ denote the least residues of $x-y^2$ and y^2+2y-x modulo p:

$$\overline{x-y^2} \equiv x - y^2 \bmod p, \quad \overline{y^2+2y-x} \equiv y^2 + 2y - x \bmod p, \quad 0 \leqslant \overline{x-y^2}, \overline{y^2+2y-x} < p.$$

If (3.6) *holds and*

$$0 \leqslant \overline{x-y^2}, \overline{y^2+2y-x} < b^{\kappa} \tag{3.8}$$

as well, then $x - y^2 = \overline{x-y^2}$, $y^2 + 2y - x = \overline{y^2+2y-x}$, and

$$y \geqslant 0 \quad \textit{and} \quad y^2 \leqslant x < (y+1)^2. \tag{3.9}$$

(c) If (3.9) *holds and $(y+1)^2 \leqslant b^{\kappa}$, then* (3.6) *and* (3.8) *both hold.*

Proof. First, note that if an integer a satisfies $h - p \leqslant a < h$, where h is any integer satisfying $0 \leqslant h \leqslant p$, then the least residue $\bar{a}$ of a modulo p satisfies

$$a = \begin{cases} \bar{a} & \text{if } 0 \leqslant \bar{a} < h \\ \bar{a} - p & \text{if } h \leqslant \bar{a} < p. \end{cases} \tag{3.10}$$

This is clear because $a \equiv \bar{a} \equiv \bar{a} - p \bmod p$, and $h - p \leqslant \bar{a} - p < 0$ if $h \leqslant \bar{a} < p$. Second, note that the inequality $h \leqslant (p+1)/2$ is equivalent to

$$h - p \leqslant 1 - h. \tag{3.11}$$

(a) If $\bar{x}, \bar{y} \in [0, b^{\kappa})$ then $x = \bar{x}$ and $y = \bar{y}$ by (3.10), since $b^{\kappa} \leqslant h \leqslant p$ by assumption (3.4). Thus, $x, y \in [0, b^{\kappa})$, and

$$-(b^{\kappa})^2 < -y^2 \leqslant x - y^2 \leqslant x < b^{\kappa},$$

while

$$-b^{\kappa} < -x \leqslant y^2 + 2y - x \leqslant y^2 + 2y < (y+1)^2 \leqslant (b^{\kappa})^2.$$

Since $b^{\kappa} \leqslant b^{2\kappa}$, we see that both $x - y^2$ and $y^2 + 2y - x$ fall within the interval $[1 - b^{2\kappa}, b^{2\kappa} - 1]$. By assumption (3.4) and inequality (3.11), this interval is contained in $[1-h, h-1] \subseteq [h-p, h-1]$.

(b) If (3.6) holds then, by (a), so does (3.7). Therefore, if (3.8) holds as well, then $x - y^2 = \overline{x-y^2}$ and $y^2 + 2y - x = \overline{y^2+2y-x}$, by (3.10) (since $b^{\kappa} \leqslant h \leqslant p$ by assumption (3.4)). In particular, $x - y^2, y^2 + 2y - x \geqslant 0$, and (3.9) follows.

(c) Note that the strict inequality $x < (y+1)^2$ is equivalent to $x \leqslant y^2 + 2y$, since x and y are integers. Thus, if (3.9) holds then $x, y \geqslant 0$,

$$0 \leqslant x - y^2 \leqslant y^2 + 2y - y^2 = 2y, \quad \text{and} \quad 0 \leqslant y^2 + 2y - x \leqslant y^2 + 2y.$$

That is, x, y, $x - y^2$, and $y^2 + 2y - x$ all fall within the interval $[0, y^2 + 2y] = [0, (y+1)^2 - 1]$, which is contained in $[0, b^{\kappa} - 1]$ if $(y+1)^2 \leqslant b^{\kappa}$. Since $b^{\kappa} \leqslant b^{2\kappa} \leqslant h \leqslant p$ (by assumption (3.4)), x, y, $x - y^2$, and $y^2 + 2y - x$ are least residues modulo p. Hence (3.6) and (3.8) both hold. □

4. THE CODE

The Rust code in this section is designed for use with the EXPANDERCOMPILERCOLLECTION (ECC) framework [5, 6], which provides a high-level interface for constructing custom arithmetic circuits over finite fields.

The verification of $y = \lfloor \sqrt{x} \rfloor$ is implemented entirely using range checks and basic field arithmetic. Specifically, we decompose and validate the inputs x, y, and two auxiliary expressions derived from x and y via their base-b representations. For modularity and clarity, we rely on the reusable utility functions `to_base_b` and `from_base_b`, discussed in detail in [1].

Algorithm 4.1 `assert_integer_sqrt`: integer square root

Require: Integers x, y, base $b \geqslant 2$, digit count $\kappa \geqslant 0$, lookup table $\mathscr{L} = \{0, 1, \ldots, b-1\}$
Ensure: Enforce $y = \lfloor \sqrt{x} \rfloor$ using base-b range checks

$x_{\text{digits}} \leftarrow \texttt{to\_base\_b}(x, b, \kappa)$
$\texttt{from\_base\_b}(x_{\text{digits}}, b, \kappa, \mathscr{L}, x)$ ▷ Range check on x
$y_{\text{digits}} \leftarrow \texttt{to\_base\_b}(y, b, \kappa)$
$\texttt{from\_base\_b}(y_{\text{digits}}, b, \kappa, \mathscr{L}, y)$ ▷ Range check on y
$y^2 \leftarrow y \times y$
$2y \leftarrow 2 \times y$
$y^2 + 2y \leftarrow y^2 + 2y$
$\delta_1 \leftarrow x - y^2$ ▷ Check $y^2 \leqslant x$
$\delta_2 \leftarrow y^2 + 2y - x$ ▷ Check $x < (y+1)^2$
$\delta_{1,\text{digits}} \leftarrow \texttt{to\_base\_b}(\delta_1, b, \kappa)$
$\texttt{from\_base\_b}(\delta_{1,\text{digits}}, b, \kappa, \mathscr{L}, \delta_1)$
$\delta_{2,\text{digits}} \leftarrow \texttt{to\_base\_b}(\delta_2, b, \kappa)$
$\texttt{from\_base\_b}(\delta_{2,\text{digits}}, b, \kappa, \mathscr{L}, \delta_2)$
return ok if all constraints are satisfied

```rust
fn assert_integer_sqrt<C: Config, API: RootAPI<C>>(
    api: &mut API,
    x: Variable,
    y: Variable,
    b: u32,
    kappa: usize,
    lookup_table: &mut LogUpSingleKeyTable,
) {
    // === Range check x ===
    let x_digits = to_base_b(api, x, b, kappa as u32);
    let _ = from_base_b(api, &x_digits, kappa, b, lookup_table, x);

    // === Range check y ===
    let y_digits = to_base_b(api, y, b, kappa as u32);
    let _ = from_base_b(api, &y_digits, kappa, b, lookup_table, y);

    // === Compute y^2 + 2y - x \& x - y^2 ===
    let y_squared = api.mul(y, y);

    let two = api.constant(2);
    let two_y = api.mul(two, y);

    let y_squared_plus_2y = api.add(y_squared, two_y);

    let diff1 = api.sub(x, y_squared);

    let diff2 = api.sub(y_squared_plus_2y, x);

    // === Range check x - y^2 ===
    let diff1_digits = to_base_b(api, diff1, b, kappa as u32);
    let _ = from_base_b(api, &diff1_digits, kappa, b, lookup_table, diff1);

    // === Range check y^2 + 2y - x ===
    let diff2_digits = to_base_b(api, diff2, b, kappa as u32);
    let _ = from_base_b(api, &diff2_digits, kappa, b, lookup_table, diff2);

    // All constraints satisfied => y = floor(sqrt(x))
}
```

Listing 1: ECC Rust API: integer square root

5. Examples

Example 5.1. Let $\tilde{x} = 0.02$ and choose a scaling factor $\alpha = 100$. Then

$$x = \lfloor \alpha^2 \tilde{x} \rfloor = \lfloor 10{,}000 \cdot 0.02 \rfloor = \lfloor 200 \rfloor = 200.$$

We wish to verify that

$$y = \lfloor \sqrt{x} \rfloor = \left\lfloor \sqrt{200} \right\rfloor = 14.$$

Off-circuit, we interpret this result as a rational approximation:

$$\sqrt{\tilde{x}} \approx \frac{y}{\alpha} = \frac{14}{100} = 0.14,$$

while the true value is $\sqrt{0.02} \approx 0.1414\ldots$, yielding a relative error of around 1%.

We now verify this relation within the arithmetic circuit. Let $p = 2^{31} - 1$ and choose $b = 10$, $\kappa = 3$, so that $b^\kappa = 10^3 = 1000$ and $b^{2\kappa} = 10^6$. Then

$$h = \frac{p+1}{2} = \frac{2^{31}}{2} = 2^{30},$$

which satisfies the assumption

$$b^{2\kappa} \leqslant h.$$

Underlying this example is the assumption that x and y lie in the interval $[h-p, h) = [-2^{30}, 2^{30})$. This ensures that their least residues modulo p can be interpreted unambiguously as signed integers: if a range check confirms that the least residue of x or y lies in $[0, h)$, then the corresponding integer is nonnegative; if it lies in $[h, p)$, then the integer is negative. In this way, the circuit's range checks on residues genuinely reflect range checks on integers.

The circuit receives $\bar{x} = 200$ and $\bar{y} = 14$ as witness values. It checks:

$$\begin{aligned}
\bar{y}^2 &= 14^2 &&= 196,\\
2\bar{y} &= 2 \cdot 14 &&= 28,\\
\bar{y}^2 + 2\bar{y} &= 196 + 28 &&= 224,\\
\overline{x - y^2} &= 200 - 196 &&= 4,\\
\overline{y^2 + 2y - x} &= 224 - 200 &&= 24.
\end{aligned}$$

Since $x, y \in [h-p, h) = [-2^{30}, 2^{30})$ and the least residues satisfy $\bar{x}, \bar{y} \in [0, h) = [0, 2^{30})$, it follows that $x = \bar{x}$ and $y = \bar{y}$. That is, their least residues coincide with their integer values. Moreover, since $x, y \in [0, b^\kappa) = [0, 1000)$ and $b^{2\kappa} \leqslant h$, it follows that both $x - y^2$ and $y^2 + 2y - x$ lie within the interval $[h-p, h)$. Combined with the fact that their least residues also lie in $[0, b^\kappa) \subseteq [0, h)$, we conclude:

$$\overline{x - y^2} = x - y^2 \quad \text{and} \quad \overline{y^2 + 2y - x} = y^2 + 2y - x.$$

This confirms that all range checks are sound and the circuit's constraints reflect valid integer relationships.

The range check details are as follows. The base-b decompositions of these values (in little-endian order) are:

- $\bar{x} = 200 \Rightarrow [0, 0, 2]$
- $\bar{y} = 14 \Rightarrow [4, 1]$
- $\overline{x - y^2} = 4 \Rightarrow [4]$
- $\overline{y^2 + 2y - x} = 24 \Rightarrow [4, 2]$

Each digit is in the lookup table $\{0, 1, \ldots, 9\}$, and the reconstructed values match their targets. Thus, the circuit verifies that

$$\bar{y} \geqslant 0 \quad \text{and} \quad \bar{y}^2 \leqslant \bar{x} < (\bar{y} + 1)^2,$$

which implies $\bar{y} = \lfloor \sqrt{\bar{x}} \rfloor$, and hence $y = \lfloor \sqrt{x} \rfloor$, as desired. ■

REFERENCES

[1] Inference Labs. *Range Check and ReLU: Arithmetic Circuit Blueprint*. https://github.com/inference-labs-inc/zkml-blueprints/blob/main/core_ops/range_check_and_relu.pdf. Accessed May 30, 2025.

[2] Inference Labs. *Range Check, Max, Min, and ReLU: Arithmetic Circuit Blueprint*. https://github.com/inference-labs-inc/zkml-blueprints/blob/main/core_ops/range_check_max_min_relu.pdf. Accessed May 30, 2025.

[3] Polyhedra Network. *Expander: Proof Backend for Layered Circuits*. https://github.com/PolyhedraZK/Expander. Accessed May 30, 2025.

[4] Polyhedra Network. *Expander: Proof System Documentation*. https://docs.polyhedra.network/expander/. Accessed May 30, 2025.

[5] Polyhedra Network. *ExpanderCompilerCollection: High-Level Circuit Compiler for the Expander Proof System*. https://github.com/PolyhedraZK/ExpanderCompilerCollection. Accessed May 30, 2025.

[6] Polyhedra Network. *ExpanderCompilerCollection: Rust Frontend Introduction*. https://docs.polyhedra.network/expander/rust/intro. Accessed May 30, 2025.