Elliptic Curve Arithmetic for Quantum Cryptanalysis: A Study of Curve25519

Yintong Luo and Craig Costello

Queensland University of Technology, Brisbane, Australia
yintong.luo@hdr.qut.edu.au craig.costello@qut.edu.au

Abstract. Recent quantum resource estimates over prime fields have largely focused on prime order elliptic curves. We implement modern affine circuit techniques on the widely deployed composite order curve, Curve25519, and obtain a scalar circuit using 47,199,530 CCX gates and 1,189 peak logical qubits. This uses approximately 35% and 18% fewer gates than Schrottenloher's published space- and gate-optimized `secp256k1` circuits, respectively. We separate three contributions. First, we apply established arithmetic and signed table techniques to Curve25519. In a comparison using the same arithmetic framework and signed half tables, its field and subgroup parameters, together with the chosen GCD iteration budgets, save 2,218,944 CCX gates relative to `secp256k1`. Second, we adapt odd signed digit recoding to quantum windows using fixed output translations. This saves a further 15,204 CCX gates and also applies to prime order curves. Third, we combine nonzero digits with a fixed 8-torsion offset point to keep the accumulator outside the lookup points' subgroup. With zero slope support and exact field arithmetic, this yields an exception-free addition schedule and nonzero denominators during slope uncomputation, without additional quantum point additions. Our counts include the full scalar schedule, lookup and cleanup. The corresponding implementation uses truncated arithmetic and heuristic GCD budgets, and excludes Fourier rotation synthesis and classical recovery.

Keywords: Quantum cryptanalysis · elliptic curves · Curve25519 · reversible arithmetic · quantum resource estimation

1 Introduction

Quantum resource estimates for elliptic curve cryptanalysis have improved significantly, but their main concrete benchmarks have been prime order curves, especially the NIST curve P-256 and `secp256k1` [PZ03, RNSL17, HJN+20, CFS26, BZG+26, Sch26]. Curve25519 has a different structure. Its field has characteristic $2^{255} - 19$, and its group has order 8ℓ for a 253-bit prime ℓ. This raises the question of how much recent affine circuits can gain from Curve25519's field and group structure.

Curve25519 underpins X25519 key exchange and Ed25519 signatures [Ber06, BDL+12, LHT16, JL17], and remains the classical component of hybrid protocols [SFG25, KS23]. We study the ECDLP in its prime order subgroup.

Recent affine constructions have substantially reduced the gate count for `secp256k1` [BZG+26, Sch26]. The ECDSA.Fail work further improves record and replay arithmetic and window selected point addition [LPH+26a, LPH+26b]. Complementary approaches target qubit usage through different arithmetic and period-finding architectures [CFS26, JL26]. Earlier Curve25519 specific studies predate these developments.[1] An updated comparison must distinguish savings due to the curve from those due to newer circuits.

[1] Huang et al. [HZH+25] benchmark Ed25519 against Häner et al.'s 2020 circuits, while Song and

Why affine short Weierstrass form? We derived and optimized reversible circuits in Montgomery and twisted Edwards form before selecting Wei25519, the short Weierstrass model of Curve25519. Quantum arithmetic costs differ from classical ECC. Schrottenloher's construction implements in-place modular multiplication by reversing the division circuit, giving comparable costs for these operations [Sch26, Section 3]. Replacing divisions with extra multiplications (as is done in typical, projective ECC implementations) therefore need not save quantum gates; this makes affine formulas with economical in-place evaluation and uncomputation particularly attractive in the quantum setting.

Twisted Edwards formulas use distinct denominators for the two output coordinates, both for complete addition and dedicated doubling [HWCD08, Section 2]. Evaluating the quotients separately requires multiple divisions; sharing an inversion introduces extra multiplications and cleanup. On the other hand, the Montgomery form offered no improvement for our windowed additions. Removing the quadratic term by the translation in (1) gives the simpler Wei25519 model, with public points from a Curve25519 ECDLP instance converted classically in advance. Affine short Weierstrass form thus gives the most efficient circuits we obtained with current arithmetic techniques.

Contributions and results. We implement the scalar computation in Shor's algorithm [Sho94] on Wei25519. To isolate curve specific effects, we also implement `secp256k1` using the same arithmetic framework with centered signed digit half tables. This controlled comparison uses 256/208-bit scalar controls and 264 GCD iterations for `secp256k1`, at a cost of 49,433,678 CCX gates (Section 5.1). Our contributions fall into three categories.

1. **Established techniques on Curve25519.** We instantiate record and replay arithmetic, pseudo-Mersenne reduction and signed half tables from prior work [Sch26, LPH+26b, HJN+20]. The smaller reduction constant and subgroup reduce arithmetic and initialization costs. At the stated control widths and GCD budgets of 261 and 264 iterations, these differences save 2,218,944 CCX gates across the full signed table circuits (Section 5.1). To separate field effects from iteration budgets, we also compare constant-point additions at 261 iterations in both fields. Wei25519 uses 66,803 fewer CCX gates and 26 fewer qubits (Table 3).

2. **A portable recoding adaptation.** Building on odd signed digit recoding [JT09] and the freedom to translate Shor's output [PZ03, Section 4.2], we recode each window independently using only X and CNOT gates. Nonzero odd digits simplify point selection and save a further 15,204 CCX gates relative to signed half tables with the same arithmetic support. This adaptation also applies to prime order curves (Section 4.2).

3. **Torsion based exception avoidance.** We apply Pornin's separation principle [Por20, Por24] to the quantum addition schedule. An order eight offset keeps the accumulator in a coset disjoint from the lookup points' prime order subgroup. Together with nonzero digits and zero slope support, this makes the addition schedule exception free under exact arithmetic and ensures nonzero denominators during slope uncomputation. The offset is absorbed into the initialization table at no additional quantum point addition cost. Proposition 1 establishes the invariant. The principle extends to other curves with suitable rational torsion.

The resulting circuit uses 47,199,530 CCX gates and 1,189 peak logical qubits. This uses approximately 35% and 18% fewer gates than Schrottenloher's published space optimized and gate optimized `secp256k1` circuits, respectively [Sch26] (Table 6). These overall

Seo [SS25] emphasize field and point operations and qubit reuse. Neither study incorporates the subsequent 2026 record and replay developments or provides the corresponding comparison with `secp256k1`.

reductions include differences in curve parameters, arithmetic and lookup choices; the controlled comparisons above isolate selected contributions.

We measure the full 253/208-bit scalar schedule, including initialization, lookup, recoding and uncomputation, using Litinski's quantum and classical trade-off [Lit23]. The implementation uses truncated arithmetic and heuristic GCD budgets. Counts exclude Fourier rotation synthesis and classical recovery, so they describe the scalar circuit rather than a complete attack.

Section 2 gives the setting; Sections 3 and 4 describe arithmetic and scalar scheduling; Section 5 evaluates resources.

2 Preliminaries

We fix the curve and subgroup notation, recall the scalar computation required by Shor's algorithm, and specify the benchmark parameters and resource conventions used throughout.

2.1 The prime order subgroup and its curve models

Curve25519, introduced by Bernstein [Ber06] in 2006, is an elliptic curve in Montgomery form defined by

$$E_M/\mathbb{F}_q\colon v^2 = u^3 + Au^2 + u, \quad \text{with} \quad q = 2^{255} - 19 \quad \text{and} \quad A = 486662.$$

Its group order is $\#E_M(\mathbb{F}_q) = 8\ell$, where ℓ is a 253-bit prime

$$\ell = 2^{252} + 27742317777372353535851937790883648493.$$

The coordinate translation

$$(x, y) = (u + A/3, v), \tag{1}$$

defines an $\mathbb{F}_q$-isomorphism from Curve25519's Montgomery form to the short-Weierstrass model Wei25519, defined as

$$E_W/\mathbb{F}_q\colon y^2 = x^3 + ax + b, \quad \text{with} \quad a = 1 - A^2/3 \quad \text{and} \quad b = 2A^3/27 - A/3.$$

We reiterate that public points in a given Curve25519 ECDLP instance are converted classically before lookup-table construction. Henceforth we assume all points are on E_W.

2.2 Shor's algorithm for the ECDLP

Let P generate the prime order subgroup of order ℓ on E_W, and let $Q = [k]P$ be a public point for an unknown scalar $k \in \mathbb{Z}/\ell\mathbb{Z}$. The ECDLP is to recover k from (P, Q). Shor's algorithm [Sho94] reduces the ECDLP to a period-finding problem. For the public points P, Q with $Q = [k]P$, the central quantum subroutine evaluates

$$f(\alpha, \beta) = [\alpha]P + [\beta]Q. \tag{2}$$

This map satisfies that $f(\alpha + k, \beta - 1) = f(\alpha, \beta)$. Its period lattice in $\mathbb{Z}^2$ is generated by $(k, -1)$ and $(\ell, 0)$.

The quantum algorithm prepares a superposition over pairs (α, β), computes $f(\alpha, \beta)$ in superposition, and then applies a quantum Fourier transform (QFT) to extract information about the hidden period. In the standard formulation, recovering k requires implementing

a reversible quantum circuit for elliptic curve scalar multiplication, namely a circuit that maps

$$|\alpha, \beta\rangle\,|\mathcal{O}\rangle \longmapsto |\alpha, \beta\rangle\,|[\alpha]P + [\beta]Q\rangle\,.$$

Our benchmark uses $0 \le \alpha < 2^{253}$ and $0 \le \beta < 2^{208}$. Following Litinski [Lit23, pp. 9–10], we retain thirteen 16-bit windows for the second scalar, giving $208 = 13 \cdot 16$ bits, and leave the remaining recovery work to classical searching. This trade-off is also used in Schrottenloher's point-addition accounting [Sch26, Section 2].

With radix 2^{16}, the first scalar occupies $\lceil 253/16 \rceil = 16$ windows and the second occupies 13. A direct lookup initializes the accumulator with the first selected point, so the remaining windows require $16 + 13 - 1 = 28$ point additions in total.

Cost conventions. We report emitted Toffoli (CCX) counts and peak live logical qubits, including controls, lookup data and arithmetic workspace across their actual lifetimes. Counts include controlled blocks even when disabled on a particular input. Each comparison identifies whether it covers a field operation, point addition or scalar schedule.

For cross-curve comparisons, we use 253/208-bit scalar controls for Wei25519 and 256/208-bit controls for `secp256k1`, with the arithmetic settings stated for different comparison.

3 Arithmetic implementation and resource optimizations

In this section we describe the field operations used in our scalar multiplication circuit and how the field and the register reuse affect our benchmark. We build on established reversible arithmetic and measure how each optimization contributes to the circuit's gate count and peak qubit usage.

3.1 The pseudo-Mersenne reduction constant and carry window

Both fields considered here have the form $p = 2^n - F$. For Curve25519, $n = 255$ and $F = 19$, and for `secp256k1`, $n = 256$ and $F = 2^{32} + 977$. An overflow beyond the n-bit register can be folded back through the identity $2^n \equiv F \pmod{p}$. The reduction constant has 5 bits in the Curve25519 prime and 33 in the `secp256k1` prime. Both fields have useful reduction structure, but the distinction is the size of the reduction constant.

In our modular-addition circuit, the main addition propagates the carry bits across the full register. An overflow, if any, is then folded back by adding F. Since F is small, adding it typically requires only short carry propagation. We exploit this by restricting the addition to the lowest L bits, reducing the gate cost, accepting a probability of error when a carry extends beyond them. The modular subtraction therefore uses the corresponding truncated borrow propagation. Higher-level circuits built from these operations inherit this property. For uniformly distributed lowest bits, the probability of such an error occurring is precisely given by

$$\Pr[\text{carry escapes the lowest } L \text{ bits}] = \frac{F}{2^L},$$

because exactly F of total 2^L lowest bits will have a carry out of 2^L. We therefore choose

$$L = \lceil \kappa + \log_2 F \rceil\,,$$

so that this fraction is at most $2^{-\kappa}$.

In our implementation, we set $\kappa = 40$, so the length of the carry window is $L = 45$ for Curve25519 and $L = 73$ for secp256k1. Shorter corrections reduce the cost of modular additions and doublings and of repeated folds inside multiplication and squaring. Compared

with `secp256k1`, Curve25519's shorter carry window saves 28 CCX gates per modular addition in our implementation.

Section 5.1 reports the resulting gate savings for individual field operations and point addition.

3.2 Lazy reduction and truncated arithmetic

Lazy reduction allows the intermediate values outside the canonical range $0, \ldots, p-1$, while within the range represented by its bit length $0, \ldots, 2^n - 1$. Avoiding an immediate canonical reduction can save operations, while short comparisons and carry windows further reduce gate counts.

Truncated comparison reduces the gate count by comparing only the most significant c bits of the operands. When the first c bits agree, the omitted bits may affect the result.

For two uniformly distributed n-bit values, the c most significant bits agree with probability 2^{-c}. When they agree, the remaining bits cause an incorrect comparison with probability at most one half. Thus,

$$\Pr[\text{incorrect comparison}] \leq 2^{-(c+1)}.$$

Such approximate arithmetic is also used in optimized quantum circuits [Sch26, LPH+26b].

In our implementation over the Curve25519 field, applying these changes successively reduces the modular-addition cost as follows:

$$1021 \xrightarrow{\text{lazy reduction}} 764 \xrightarrow{\text{comparison: } c=40} 549 \xrightarrow{\text{carry window: } L=45} 339 \quad \text{CCX}.$$

Lazy reduction reduces the modular-addition cost from 1,021 to 764 CCX. Truncating the comparison to $c = 40$ lowers it to 549 CCX, and restricting the carry window to $L = 45$ reduces it further to 339 CCX.

3.3 GCD history and replay

Following Schrottenloher [Sch26, Section 3], we implement in-place multiplication and division over $\mathbb{F}_p$ for the point addition circuit, i.e.

$$|x\rangle\,|y\rangle \longmapsto |x\rangle\,|xy\rangle \qquad \text{and} \qquad |x\rangle\,|y\rangle \longmapsto |x\rangle\,\left|yx^{-1}\right\rangle,$$

for $x \neq 0$. Multiplication and division are implemented by reversibly converting the $|x\rangle$ register into a GCD history. The circuit runs the binary GCD algorithm on (p, x) and records the decisions controlling its swaps, subtractions and halvings. These decisions form the GCD tape, also called a dialog [KSG+25, Sch26]. For multiplication, we take the GCD tape and apply the Bézout reconstruction circuit to reconstruct the value of x into $|y\rangle$. For division, we run the same circuit but in reverse, meaning reverse the order of the operations and apply their inverses.

We adopt the jump-2 organization described by Long et al. [LPH+26b, Section 4.2]. Start by initializing $u = p$ and $v = x$. If x is odd, we perform $v = p - x$, and now we start with an odd u and an even v. Each iteration halves v, halves it again if it is still even, and then performs the required comparison, swap and subtraction. The resulting decisions have only five possible combinations, so each iteration can be encoded as one base-5 digit with three digits that can fit in seven qubits. For a fixed iteration count I divisible by three, the packed GCD history occupies $7I/3$ qubits.

As the GCD computation progresses, the operands u and v shrink. The implementation reuses released qubits to store the growing GCD history. To determine when these qubits can be reused, we estimate how many bits the operands still require at each iteration. Releasing qubits too early would overwrite bits that still carry operand information.

Heuristic 1 (GCD operand widths)**.** At iteration i, starting from $i = 0$, choose the width of each operand register according to

$$n - 1.062i + \Delta, \qquad \text{with} \qquad \Delta = \lceil 2.3\sqrt{n} \rceil,$$

rounding upward to a multiple of two in the original n-bit register width. The coefficient 1.062 estimates the decrease in the bit length of $\max(u, v)$ per jump-2 iteration (See Appendix A). The additional margin Δ accommodates variation between inputs [Sch26, Section 3.1]

Our heuristic is also consistent with the empirical results reported in ECDSA.Fail [LPH+26a, Section 5.3.7].

At each iteration, the comparison uses the most significant c bits of the remaining operand registers. As the register widths decrease according to the schedule, this comparison window moves toward the lower bit positions. Its width is c until fewer than c bits remain, at which point all remaining bits are compared. Since the margin Δ can introduce leading zeros shared by both operands, we choose c relative to this margin.

Heuristic 2 (GCD comparison width)**.** Let w_i be the operand-register width at iteration i, and let Δ be the margin in the width schedule from Heuristic 1. We compare the most significant c_i bits of each operand, where

$$c_i = \min\{w_i, c\} \qquad \text{with} \qquad c = \Delta + \tau,$$

and where τ is an additional comparison parameter.

Our implementation uses $\tau = 63$, giving $c = 100$ for $n = 255$. Once $w_i \leq c$, the comparison uses all remaining operand bits.

After recording the GCD history, the circuit replays it to perform the in-place multiplication. The replay leaves a temporary register containing p if the product is nonzero and 0 otherwise, so its cleanup must distinguish these cases. Let v denote the result of the in-place multiplication, and let $t \in \{0, p\}$ be the temporary value and define

$$f(t) = \begin{cases} 0, & t = 0, \\ 1, & t = p. \end{cases}$$

The flag $f(t)$ controls a bitwise XOR with the constant p, clearing the temporary register:

$$|f(t)\rangle\,|t\rangle \longmapsto |f(t)\rangle\,|t \oplus f(t)p\rangle = |f(t)\rangle\,|0\rangle\,.$$

Since $f(t) = [v \neq 0]$, the flag is computed and uncomputed using the retained multiplication result v. Division reverses this process, using the numerator to prepare the temporary register before replay. The zero test assumes canonical inputs, with zero represented by the integer 0.

3.4 A heuristic iteration budget

Unlike classically, the GCD circuit must decide the number of iterations before the input-dependent decisions are known. More iterations accommodate longer GCD computations, but also increase the gate count and the history storage.

There is a simple intuition for the jump-2 iteration count. An ordinary binary-GCD iteration performs one halving. A regular jump-2 iteration performs one halving and a second whenever the first quotient is even [LPH+26b, Section 4.2]. If that second halving occurs about half the time, each jump-2 iteration performs about 1.5 halvings. Dividing the ordinary-step estimate of approximately $1.413n$ iterations [Sch26, Section 3.1] by 1.5

gives $1.413n/1.5 \approx 0.942n$. This explains the approximate scale of the observed count, without establishing its distribution or an upper bound.

The empirical estimates $0.94263n$ for the mean and $0.31031\sqrt{n}$ for the standard deviation can be validated using the script mentioned in Appendix A, based on the iteration counts of 3000 random elements of $\mathbb{F}_q$ to obtain the following heuristic.

Heuristic 3 (Jump-2 iteration count)**.** We choose the iteration count as

$$I(n) = 3\left\lceil \frac{0.94263n + z\,0.31031\sqrt{n}}{3} \right\rceil, \qquad z = 2.4/0.5974 \approx 4.017. \tag{3}$$

Rounding upward to a multiple of three allows the GCD history to be packed in groups of three iterations.

We choose $z \approx 4.017$ to express the ordinary binary-GCD allowance of $2.4\sqrt{n}$ in units of its empirical standard deviation, $0.5974\sqrt{n}$.

Evaluating Equation (3) at the two field sizes gives $I(255) = 261$ and $I(256) = 264$, similar to the iteration budget reported by Long et al. [LPH+26b, Section 4.2].

Note that the heuristic does not guarantee termination for every input or equal failure probabilities in the two fields. We therefore report both the default iteration counts and a comparison with 261 iterations in each field in Section 5.1.

3.5 Resource-optimized square subtraction

The point-addition circuit requires the field operation

$$|x\rangle\,|z\rangle \longmapsto |x\rangle\,\left|z - x^2\right\rangle,$$

which preserves the slope while subtracting its square from the other coordinate register. The circuit accumulates the square directly into the destination register, following the approach used by Schrottenloher [Sch26].

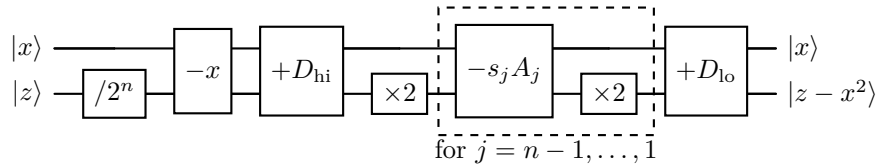

Figure 1: Arithmetic circuit for square subtraction. Each wire represents an n-qubit register. The two-register gates preserve x and update z by the indicated addition or subtraction and all arithmetic is modulo p. Here $A_j = x \bmod 2^j$, $s_j = 2x_j - 1$, and $D = \sum_{j=0}^{n-1} x_j 2^{2j} = D_{\mathrm{lo}} + 2^n D_{\mathrm{hi}}$, with $0 \le D_{\mathrm{lo}} < 2^n$. The dashed group is repeated for $j = n-1, \ldots, 1$, giving n doublings in total. The gate $/2^n$ denotes n modular halvings, which compensate for the effect of the doublings on the input accumulator. Here $n = 255$ for Curve25519 and $n = 256$ for `secp256k1`. Temporary registers are omitted.

Figure 1 shows the square-subtraction circuit. Table 1 separates the measured costs. Both circuits use chunked additions for the diagonal terms. Curve25519 uses 16,183 fewer CCX gates, a reduction of 18.97%.

The largest saving comes from modular doubling and halving. Each costs 45 CCX gates over the Curve25519 field and 73 over the `secp256k1` field, reflecting the shorter carry window used when folding by $F = 19$. The circuit performs n halvings and n doublings, giving a combined saving of 14,426 CCX gates. Each row adds or subtracts a j-bit value A_j from the n-bit accumulator. The circuit operates on the lowest $j + \kappa$ bits, allowing κ additional bits for carry propagation. When $j + \kappa \ge n$, it uses full-width modular arithmetic. Both circuits use $\kappa = 40$, while the modular-reduction window is $L = 45$ for Curve25519 and $L = 73$ for `secp256k1`.

Table 1: Square-subtraction costs with $\kappa = 40$ in both fields. All entries are CCX counts.

Component	Wei25519	`secp256k1`	Saving
Initial modular halvings	11,475	18,688	7,213
Modular doublings	11,475	18,688	7,213
Triangular-row additions	44,911	46,325	1,414
Initial subtraction and diagonal additions	1,273	1,616	343
Total	69,134	85,317	16,183

4 Organizing scalar multiplication

The scalar circuit adds points selected from classically precomputed tables. In Sections 4.1–4.2, we reduce the lookup cost by using signed digits to halve the table size, then odd-only digits to simplify the point-selection circuit. Signed half-tables were already used in quantum scalar multiplication by Häner et al. [HJN+20], but the later resource estimates of Schrottenloher [Sch26] and Chevignard et al. [CFS26] use full window tables without this reduction. We measure the savings across the complete scalar multiplication circuit, including recoding, sign handling, and uncomputation.

In Section 4.3, initializing the accumulator with a fixed torsion component then prevents the accumulator and the updated accumulator from sharing the x-coordinate with the lookup point A. We show that the addition schedule is exception-free despite using an incomplete Weierstrass addition formula.

4.1 Window tables and signed digits

We start by splitting α and β into windows of width w, i.e.

$$\alpha = \sum_i \alpha_i 2^{wi} \qquad \text{and} \qquad \beta = \sum_j \beta_j 2^{wj}.$$

Recall that for our implementation, $w = 16$. Define the base points for these windows as

$$P_i = [2^{wi}]P \qquad \text{and} \qquad Q_j = [2^{wj}]Q.$$

The scalar multiplication can then be written as:

$$T(\alpha, \beta) = [\alpha]P + [\beta]Q = \sum_i [\alpha_i]P_i + \sum_j [\beta_j]Q_j.$$

To halve the lookup tables, we center each window value around 0, by taking

$$\widetilde{\alpha}_i = \alpha_i - 2^{w_i - 1} \qquad \text{and} \qquad \widetilde{\beta}_j = \beta_j - 2^{w_j - 1},$$

where $w_i = w_j = w$ except when the most significant window is shorter. A centered w_i-bit digit lies in the range:

$$[-2^{w_i-1},\, 2^{w_i-1} - 1].$$

On a Weierstrass curve, a point and its inverse share the same x-coordinate. We therefore precompute only the positive multiples

$$P_i,\, [2]P_i,\, [3]P_i,\, \ldots,\, [2^{w_i-1}]P_i,$$

and similarly for Q_j. A zero digit disables the addition. Each table has 2^{w_i-1} entries, half the number of possible window values.

Using the centered digits changes the computed point to

$$\sum_i [\widetilde{\alpha}_i]P_i + \sum_j [\widetilde{\beta}_j]Q_j = T(\alpha, \beta) - K,$$

where

$$K = \sum_i [2^{w_i-1}]P_i + \sum_j [2^{w_j-1}]Q_j.$$

The offset K is independent of α and β and therefore preserves the Fourier-sampling distribution, as also observed by Häner et al. [HJN+20, Section 5.1].

To add a negative digit, the circuit negates the accumulator R before and after adding the selected positive multiple A, i.e.

$$-\big((-R) + A\big) = R - A.$$

Address zero stores $[2^{w_i-1}]P_i$ and is shared by the digits 0 and -2^{w_i-1}, with the sign bit distinguishing these cases: 0 for no update, and 1 for adding the negative endpoint.

4.2 Odd-only digits

We adapt the odd signed-digit recoding of Joye and Tunstall [JT09, Section 3.2], with a few modifications to the quantum setting. Recall from the previous section that Shor's algorithm permits a fixed output translation without changing the Fourier-sampling distribution. We use this freedom to convert each binary window independently, without propagating adjustments between windows. This independent conversion requires only X and CNOT gates, avoiding the recursive scalar updates and their reversible cleanup.

For the i-th window, define the odd digits and their corresponding half-bases by

$$\widehat{\alpha}_i = 2\widetilde{\alpha}_i + 1, \qquad \widehat{\beta}_j = 2\widetilde{\beta}_j + 1,$$

$$H_i^{(P)} = [2^{-1} \bmod \ell]P_i, \qquad \text{and} \qquad H_j^{(Q)} = [2^{-1} \bmod \ell]Q_j.$$

These points are computed classically. Since

$$[\widehat{\alpha}_i]H_i^{(P)} = [\widetilde{\alpha}_i]P_i + H_i^{(P)} \qquad \text{and} \qquad [\widehat{\beta}_j]H_j^{(Q)} = [\widetilde{\beta}_j]Q_j + H_j^{(Q)},$$

the odd-only encoding changes the centered result by a fixed translation, independent of α and β.

For each window, precompute the positive odd multiples

$$H_i^{(P)}, [3]H_i^{(P)}, [5]H_i^{(P)}, \ldots, [(2^{w_i} - 1)]H_i^{(P)}, \qquad \text{and}$$
$$H_j^{(Q)}, [3]H_j^{(Q)}, [5]H_j^{(Q)}, \ldots, [(2^{w_j} - 1)]H_j^{(Q)},$$

for α_i and β_j.

All selected points are nonzero because the half-bases have order ℓ and the positive multipliers are smaller than ℓ. A 16-bit window still requires 32,768 table entries.

To obtain the sign and table address for α_i, let a_i contain its lowest $w_i - 1$ bits. Flipping the most significant bit of α_i gives the sign bit s_i, which is zero for a positive digit and one for a negative digit. The table address h_i is

$$h_i = \begin{cases} a_i, & s_i = 0, \\ 2^{w_i-1} - 1 - a_i, & s_i = 1, \end{cases} \qquad \widehat{\alpha}_i = (-1)^{s_i}(2h_i + 1),$$

and similarly for β_j.

Thus the signs and addresses are computed using only X and CNOT gates. There is no zero digit or special negative endpoint, and every update is enabled. This removes the zero/endpoint controls and the controlled copy used by square subtraction. Section 5.2 measures these savings together with the reduction in recoding cost.

4.3 Using torsion to prevent exceptional additions

Fixed accumulator offsets have previously been used in quantum ECDLP algorithms. Proos and Zalka [PZ03, Section 4.2] observed that initializing the accumulator at a fixed nonzero multiple of the base point leaves the Fourier-sampling distribution unchanged, and used random initialization within the prime-order subgroup to bound the contribution of exceptional additions. We use the same freedom to translate the output, choosing an offset outside the prime-order subgroup to exclude exceptional additions throughout the computation.

Our construction applies the torsion-based separation principle used by Pornin [Por20, Por24], who translates addition inputs by a point of order two to place them in disjoint cosets. Here, we initialize the accumulator in $T_0 + G$, where T_0 has order eight, while all lookup points lie in G. Assuming exact field arithmetic, this invariant ensures that both the accumulator and its updated value have an x-coordinate distinct from that of the nonzero lookup point. Consequently, the denominators required for point addition and slope uncomputation remain nonzero. Together with the zero-slope support of Section 3.3, this yields an exception-free addition schedule while retaining the incomplete affine formulas. The following proposition establishes the required invariant.

Proposition 1 (Torsion offset)**.** *Let G be the prime-order subgroup of Wei25519 of order ℓ, and let T_0 be a point of order 8. If the accumulator is initialized in $T_0 + G$, then adding or subtracting points from G keeps the accumulator in the coset $T_0 + G$. Moreover, for every accumulator $R \in T_0 + G$ and nonzero point $A \in G$, both R and $R + A$ are finite and satisfy*

$$x(R) \neq x(A), \qquad x(R + A) \neq x(A).$$

Thus, the slope denominator and the factor used to erase the slope are nonzero throughout the addition schedule. Both accumulator points also have nonzero y-coordinates. The same statements hold with T_0 replaced by $-T_0$ during sign handling.

Proof. Since $ord(T_0) = 8$ and G has odd prime order ℓ, we have $T_0 \notin G$. Hence the coset $T_0 + G$ is disjoint from G. For an accumulator $R = T_0 + B$, where $B \in G$, and any $A \in G$,

$$R \pm A = T_0 + (B \pm A) \in T_0 + G.$$

Thus every addition or subtraction preserves the coset.

Since $\mathcal{O} \in G$, both R and $R + A$ lie in $T_0 + G$, so neither point is $\mathcal{O}$. On any Weierstrass curve, two finite points with the same x-coordinate are equal or negatives of each other. Therefore,

$$x(R) = x(A) \implies R = \pm A,$$

which is impossible because $\pm A \in G$. Similarly,

$$x(R + A) = x(A) \implies R + A = \pm A \implies R = \mathcal{O} \text{ or } R = -2A.$$

Both possibilities are excluded because $\mathcal{O}, -2A \in G$. Consequently, the slope denominator $x(R) - x(A)$ and the cleanup factor $x(R + A) - x(A)$ are nonzero.

For any $R \in T_0 + G$, $R = T_0 + B$ for some $B \in G$. Then, the point $[\ell]R = [\ell]B + [\ell]T_0 = [\ell]T_0$ has order 8. Hence R cannot have order 2, which excludes $y(R) = 0$. This also applies to $R + A$. The same argument applies in $-T_0 + G$ during sign handling. □

In our implementation, we add T_0 to every entry of the first-window lookup table. This initializes the accumulator in $T_0 + G$ without an additional quantum point addition.

Overall, combining the odd-only encoding offset $-K + \sum_i H_i^{(P)} + \sum_j H_j^{(Q)}$, and the torsion offset T_0, the final accumulator output is:

$$R_{\text{final}} = [\alpha]P + [\beta]Q - K + \sum_i H_i^{(P)} + \sum_j H_j^{(Q)} + T_0.$$

The total offset is independent of α and β, so as discussed previously, it preserves the Fourier-sampling distribution.

The torsion offset ensures that the slope denominator is nonzero. However, $y(R) = y(A)$ can still give a zero slope. The zero input support described in Section 3.3 handles this case.

4.4 The complete scalar multiplication circuit

We now put everything together. Algorithm 1 combines the signed half-tables, odd-only encoding, torsion initialization and zero-slope support described above. It adapts the affine addition schedule of Schrottenloher [Sch26] to the complete scalar multiplication.

For a classically precomputed table $\mathcal{T}$, Lookup(h) takes an address h stored in a quantum register and writes the selected entry into zero temporary registers:

$$|h\rangle\,|0\rangle \longmapsto |h\rangle\,|\mathcal{T}[h]\rangle\,.$$

The corresponding Unlookup(h, a) takes the address h and the temporary register a holding $\mathcal{T}[h]$, and clears a:

$$|h\rangle\,|\mathcal{T}[h]\rangle \longmapsto |h\rangle\,|0\rangle\,.$$

Both operations preserve the address register. Unlookup includes the phase correction required by measurement-based erasure. The table entries are specified in the algorithm comments.

OddRecode denotes the reversible conversion from a binary window to its sign and table address described in Section 4.2. Its inverse restores the window's original binary encoding.

Let N and M denote the numbers of windows in α and β, respectively. The circuit processes each scalar from its most significant window to its least significant window, beginning with α.

For the 253/208 configuration, the circuit processes 16 windows of α followed by 13 windows of β, from most to least significant within each scalar.

The first window α_{N-1} contains 13 bits and initializes the accumulator through an 8,192-entry table containing the torsion offset. The remaining 28 windows each contain 16 bits and use a positive half-table with 32,768 entries.

After each addition, the circuit reverses the digit conversion to restore the window's original binary encoding so that the subsequent QFT acts on the original scalar encoding.

Loaded table values are erased before division and multiplication, allowing their qubits to be reused for the GCD workspace.

Section 5.2 evaluates the gate count and peak qubit width of the design choices in this section, including initialization, recoding, sign handling and uncomputation.

5 Resource evaluation and comparison

Our complete Wei25519 scalar circuit uses 47,199,530 CCX gates ($\approx 2^{25.49}$) and 1189 peak logical qubits. We first measure the advantages of the Curve25519 field and subgroup, then the contributions of signed half-tables and odd-only encoding.

All scalar circuits use radix 2^{16}, one initialization lookup and 28 point additions. The control widths are 253/208 for Wei25519 and 256/208 for the `secp256k1`. Gate counts and peak qubit usage are measured over the complete circuits.

5.1 Curve25519-specific advantages

Field arithmetic. The smaller reduction constant lowers the cost of the operations in Table 2. Modular addition saves 29 CCX gates: 28 from the shorter reduction window

Algorithm 1 Odd-only scalar multiplication with signed half-tables and torsion initialization

Input: Scalar windows $\alpha_0, \ldots, \alpha_{N-1}$ and $\beta_0, \ldots, \beta_{M-1}$, zero coordinate registers (x, y), and the classically precomputed tables.

Output under exact arithmetic: Coordinates of R_{final}, with the scalar registers unchanged and all ancillas returned to $|0\rangle$.

Field arithmetic is modulo p, $F = 2^n - p$, and $\overline{x} = 2^n - 1 - x$.

procedure ScalarMultiply
  $(x, y) \leftarrow \text{Lookup}(\alpha_{N-1})$ ▷ initialize (x, y) with $T_0 + [\widehat{\alpha}_{N-1}]H_{N-1}^{(P)}$
  for $i = N-2, \ldots, 0$ **do**
    OddAdd$(x, y, \alpha_i, H_i^{(P)})$
  end for
  for $j = M-1, \ldots, 0$ **do**
    OddAdd$(x, y, \beta_j, H_j^{(Q)})$
  end for
end procedure

procedure OddAdd$(x, y, \alpha_i, H_i^{(P)})$
  $(s_i, h_i) \leftarrow \text{OddRecode}(\alpha_i)$
The selected positive point is $A = [2h_i + 1]H_i^{(P)} = (x_A, y_A)$.
  if $s_i = 1$ **then** $y \leftarrow -y$
  $(a, b) \leftarrow \text{Lookup}(h_i)$ ▷ First lookup, (x_A, y_A)
  $(x, y) \leftarrow (x - a,\ y - b)$
  $\text{Unlookup}(h_i, a, b)$ ▷ Clear the loaded (x_A, y_A)
  $y \leftarrow yx^{-1}$ ▷ In-place division
  $x \leftarrow x - y^2$ ▷ Uncontrolled square subtraction
  $x \leftarrow \overline{x}$
  $a \leftarrow \text{Lookup}(h_i)$ ▷ Second lookup, $k_A = (F - 1 + 3x_A) \bmod p$
  $x \leftarrow x - a$
  $\text{Unlookup}(h_i, a)$ ▷ Clear the loaded k_A
  $y \leftarrow xy$ ▷ In-place multiplication
  $(a, b) \leftarrow \text{Lookup}(h_i)$ ▷ Third lookup, $(x_A, y_A) \bmod p$
  $x \leftarrow x + a$
  $y \leftarrow y + b$
  $\text{Unlookup}(h_i, a, b)$ ▷ Clear the loaded coordinates
  $y \leftarrow -y$
  if $s_i = 1$ **then** $y \leftarrow -y$
  $\alpha_i \leftarrow \text{OddRecode}^{-1}(s_i, h_i)$
end procedure

and 1 from the smaller field size. Modular doubling saves 28 CCX gates. These savings accumulate in the square, multiplication and division circuits. Both fields use a $\kappa = 40$ carry margin for narrow integer rows in the square. Their modular-reduction windows L remain 45 and 73 bits, respectively, as described in Section 3.1.

Table 2: Field operation costs with the same 40-bit carry margin in both square circuits (in CCX)

Operation	Wei25519	`secp256k1`	Saving
Modular addition	339	368	29
Modular doubling	45	73	28
Multiplication by F	2,299	4,495	2,196
Square subtraction	69,134	85,317	16,183

Previous work benchmarks point addition arithmetic separately from table lookup, as in Schrottenloher [Sch26, Table 1] and the original ECDSA.Fail challenge [LPH+26a, Section 5.2.1]. Following this approach, Table 3 compares the same constant-point addition construction over the Curve25519 and `secp256k1` fields. This comparison excludes lookup costs to measure the savings from field arithmetic and the GCD iteration budget.

From Heuristic 3, we have a GCD iteration budget of 261 for Wei25519 and 264 for `secp256k1`. At these settings, Wei25519 saves 77,233 CCX gates and 33 qubits. Holding both iteration counts at 261 leaves a saving of 66,803 CCX gates and 26 qubits, coming from 1 bit field size difference and cheaper modular reduction.

Table 3: Constant-point addition costs. Each resource pair is CCX / peak qubits. Percentages are relative to `secp256k1`.

GCD iterations	Wei25519	`secp256k1`	Saving
261 / 264	1,580,778 / 1,173	1,658,011 / 1,206	4.66%
261 / 261	1,580,778 / 1,173	1,647,581 / 1,199	4.05%

At the 261/264 iteration counts, division and multiplication contribute 30,441 and 30,435 CCX to the saving. Square subtraction contributes 16,183 and the six constant subtractions contribute 174. The three additional GCD iterations used for `secp256k1` also require seven more history qubits.

At 261 iterations Wei25519 saves 2 qubits from its two field register. During GCD replay, the chunked adder requires 49 temporary qubits in both fields, while modular reduction requires 45 for Wei25519, which does not exceed 49, and 73 for `secp256k1`. Since the larger requirement determines its peak, the total saving is therefore

$$2 \cdot (256 - 255) + (73 - 49) = 26 \text{ qubits.}$$

Subgroup size and initialization. We use a 253-bit first scalar control, matching the bit length of the subgroup order ℓ. The second control remains 208 bits, following the trade-off discussed by Litinski [Lit23, pp. 9–10]. This gives a 13-bit first window for Wei25519, and 16-bit for `secp256k1`. The initialization tables therefore have 8,192 and 65,536 entries, respectively. This also aligns with their lookup costs, 8,190 and 65,534 CCX gates respectively, which gives a saving of

$$65,536 - 8,192 = 57,344 \text{ CCX.}$$

for the first point lookup. This saving occurs once per scalar circuit. After initialization, both circuits process 15 remaining α windows and 13 β windows, giving 28 point additions.

Each window has 16 bits and uses a signed half-table with $2^{15} = 32,768$ entries. The smaller initialization table reduces the gate count but does not reduce peak qubit usage, which is determined by the subsequent point additions with 16-bit controls.

Combined field and subgroup savings. Using centered signed digits for both curves, the full-circuit difference separates into initialization and point additions as

$$\underbrace{57,344}_{\text{Initialization}} + \underbrace{28 \cdot (1,763,148 - 1,685,948)}_{\text{Point additions}} = 2,218,944\,\text{CCX}.$$

Thus for the same signed construction, Wei25519 uses 4.49% fewer CCX gates in the stated allocations and arithmetic settings, compared to `secp256k1`. The arithmetic term includes the field parameters and the different GCD iteration counts (261/264).

Torsion initialization. Adding T_0 to the initialization window entries keeps the accumulator outside the prime-order subgroup, as shown in Proposition 1. Assuming correct arithmetic, the resulting addition schedule is exception-free. This requires no additional lookup table entries or quantum point additions. The group of `secp256k1` has prime order, so this advantage is Curve25519 specific.

5.2 Portable lookup and encoding improvements

In the previous subsection we discussed the structural advantages for Curve25519. In this subsection, we summarize the design choices that also apply to other curves, and measure their contributions to the circuit cost.

Table 4 compares unsigned tables, signed half-tables and the complete odd-only construction at the same 253/208 control widths. The counts include initialization, recoding, sign handling and uncomputation. The signed and odd-only circuits use the same zero-input support for multiplication and division, described in Section 3.3. The unsigned reference retains its original arithmetic. The complete odd-only construction follows Algorithm 1.

Table 4: Effect of lookup and encoding choices on the complete Wei25519 scalar circuit.

Construction	CCX per addition	Total CCX
Unsigned tables	1,780,046	49,849,478
Signed half-tables	1,685,948	47,214,734
Complete odd-only	1,685,405	47,199,530

Signed half-tables. A 16-bit unsigned window has 65,536 entries, whereas a signed half-table has 32,768. Each of the three forward lookup stages saves 32,768 CCX gates, and each corresponding unlookup saves 128 CCX gates. Including recoding, sign handling and the additional arithmetic support, the signed construction uses 2,634,744 fewer CCX gates (5.29%) than the unsigned reference.

Odd-only encoding. The odd digits retain the half-table size and simplify point selection. With multiplication, division and coordinate restoration held fixed, odd-only encoding saves 543 CCX gates per addition. Table 5 gives the breakdown. Across the 28 additions, the saving is

$$28 \cdot 543 = 15,204\,\text{CCX},$$

or 0.0322% of the signed circuit, with no increase in peak width.

Table 5: CCX savings from odd-only encoding per 16-bit addition compared to signed half-tables with the same arithmetic support.

Work saved	CCX saved
Centered recoding and its inverse	60
Zero-digit enable logic	116
Negative-endpoint correction	112
Controlled square-source copy	255
Total	543

The complete odd-only circuit saves 2,649,948 CCX gates (5.32%) over the unsigned reference. All three constructions peak at 1189 qubits.

Signed half-tables and odd-only encoding do not depend on the Curve25519 reduction constant or its torsion subgroup, and can also be used over `secp256k1`. The torsion initialization is a separate feature of the Wei25519 construction.

5.3 Overall comparison

Table 6 compares the complete Wei25519 scalar circuit with the space-optimized and gate-optimized `secp256k1` constructions reported by Schrottenloher [Sch26, Table 2].

Table 6: Overall resource comparison with published `secp256k1` estimates.

Construction	Toffoli count	Logical qubits
`secp256k1`, space-optimized [Sch26]	$2^{26.11}$	1,208
`secp256k1`, gate-optimized [Sch26]	$2^{25.78}$	1,462
Wei25519, this work	$47,199,530 \approx 2^{25.49}$	1,189

The Wei25519 circuit uses approximately 35% fewer gates and 19 fewer qubits than the space-optimized construction. Compared with the gate-optimized construction, it uses approximately 18% fewer gates and 273 fewer qubits. These reductions reflect the curve parameters, the arithmetic, and scalar-encoding choices used in the respective circuits.

Schrottenloher [Sch26, Tables 1–2] estimates the total cost from 28 point additions, each requiring three lookups of 2^{16} entries. The space-optimized and gate-optimized estimates break downs are, respectively,

$$28 \cdot \Big(\underbrace{2^{21.19}}_{\text{Point addition}} + \underbrace{3 \cdot 2^{16}}_{\text{Lookups}} \Big) \approx 2^{26.11}, \quad \text{and}$$
$$28 \cdot \Big(\underbrace{2^{20.83}}_{\text{Point addition}} + \underbrace{3 \cdot 2^{16}}_{\text{Lookups}} \Big) \approx 2^{25.78}.$$

The ECDSA.Fail challenge benchmarks individual constant-point additions, with one point supplied classically. Its scoring uses peak qubit usage and average executed Toffoli counts, and permits nonce search to select validation inputs [LPH+26a, Sections 3.1–3.3 and 6]. Here, we measure the complete scalar multiplication circuit, including initialization, table lookup, recoding and uncomputation, without nonce search. The subsequent windowed constructions provide independent validation but still evaluate individual point additions [LPH+26b, Sections 6.4 and 7]. We therefore do not extrapolate these results into complete scalar circuit totals or include them in the numerical comparison.

Our focus is on the arithmetic advantages of the Curve25519 field and structural optimizations in table lookup, scalar encoding and accumulator initialization. These

advantages and optimizations can be combined with future improvements to individual stages of the point addition circuit. Evaluating these combinations and measuring the resulting resource savings is left to future work.

6 Conclusion

In this work, we explore different techniques to reduce the cost of quantum circuit to compute Curve25519 scalar multiplication. The smaller reduction constant and subgroup order lower arithmetic and initialization costs, while torsion initialization gives an exception-free addition schedule under exact field arithmetic. Signed half-tables and odd-only encoding provide portable reductions in lookup and point-selection costs. The complete Wei25519 circuit with 253/208-bit scalar controls uses 47,199,530 CCX gates and 1,189 logical qubits.

Future work should focus on optimizing division and in-place multiplication, as they account for approximately 95.5% of the gates in the addition circuit. Future improvements can be combined with the techniques presented here to further reduce scalar-multiplication costs. Under comparable constructions, we expect Curve25519 to retain the advantage over `secp256k1`.

References

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie–Hellman speed records. In *Public Key Cryptography – PKC 2006*, pages 207–228. Springer, 2006.

[BZG+26] Ryan Babbush, Adam Zalcman, Craig Gidney, Michael Broughton, Tanuj Khattar, Hartmut Neven, Thiago Bergamaschi, Justin Drake, and Dan Boneh. Securing elliptic curve cryptocurrencies against quantum vulnerabilities: Resource estimates and mitigations. arXiv:2603.28846v2, 2026.

[CFS26] Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher. Reducing the number of qubits in quantum discrete logarithms on elliptic curves. In *Advances in Cryptology – EUROCRYPT 2026*, pages 371–401. Springer, 2026. Full version: Cryptology ePrint Archive, Paper 2026/280, revised 4 June 2026.

[HJN+20] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. Improved quantum circuits for elliptic curve discrete logarithms. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 425–444, Cham, 2020. Springer International Publishing.

[HWCD08] Hüseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted edwards curves revisited. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2008.

[HZH+25] Yan Huang, Fangguo Zhang, Zhi Hu, Zijian Zhou, and Longjiang Qu. Improved quantum circuits for elliptic curve discrete logarithm problems on Ed25519. *Quantum Information Processing*, 24:311, 2025. Article 311; published 23 September 2025.

[JL17] Simon Josefsson and Ilari Liusvaara. Edwards-curve digital signature algorithm (EdDSA). RFC 8032, 2017.

[JL26] Sunghyeon Jo and Gye Jin Lee. Quantum discrete logarithms on elliptic curves with $5n/2 + o(n)$ logical qubits and $\widetilde{O}(n^2)$ Toffoli gates. Cryptology ePrint Archive, Paper 2026/2014, 2026.

[JT09] Marc Joye and Michael Tunstall. Exponent recoding and regular exponentiation algorithms. In Bart Preneel, editor, *Progress in Cryptology – AFRICACRYPT 2009*, pages 334–349, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[KS23] Ehren Kret and Rolfe Schmidt. The PQXDH key agreement protocol, 2023.

[KSG+25] Tanuj Khattar, Noah Shutty, Craig Gidney, Adam Zalcman, Noureldin Yosri, Dmitri Maslov, Ryan Babbush, and Stephen P. Jordan. Verifiable quantum advantage via optimized DQI circuits. arXiv:2510.10967, 2025.

[LHT16] Adam Langley, Mike Hamburg, and Sean Turner. Elliptic curves for security. RFC 7748, 2016.

[Lit23] Daniel Litinski. How to compute a 256-bit elliptic curve private key with only 50 million Toffoli gates. arXiv:2306.08585, 2023.

[LPH+26a] Jieyi Long, Theodore Pender, Zhao Huang, Manuel B. Santos, Samrendra Kumar Singh, Bartosz Naskręcki, Bit Wonka, Pierre-Luc Dallaire-Demers, Francesco Giannicola, Ruben M. L. Paschoarelli, Oli Freuler, Jackie Chia-Hsun Lee, Vasily Gnuchev, Gopi Kannappan, John Boyer, Xavier Butler, Akash Balasubramani, Jordan Newman, Bereket Dereje, Alexander Hertlein, Robert Kodra, Lucas Levy, Shaan Patel, JT Rose, Matt Zweil, Okechukwu Wisdom, Tarek El-Eter, Edison Lee, Michael Dong, Alan Li, Anto Joseph, Gajesh Naik, Gautham Anant, Soubhik Deb, and Justin Drake. ECDSA.Fail: Open autoresearch for optimizing elliptic-curve point addition in Shor's algorithm. arXiv:2609.09582v2 [quant-ph], 2026. Version 2, 19 September 2026.

[LPH+26b] Jieyi Long, Theodore Pender, Zhao Huang, Manuel B. Santos, Samrendra Kumar Singh, Bartosz Naskręcki, BitWonka, Pierre-Luc Dallaire-Demers, Francesco Giannicola, Ruben M. L. Paschoarelli, Oli Freuler, Jackie Chia-Hsun Lee, Vasily Gnuchev, Gopi Kannappan, John Boyer, Xavier Butler, Akash Balasubramani, Jordan Newman, Bereket Dereje, Alexander Hertlein, Robert Kodra, Lucas Levy, Shaan Patel, JT Rose, Matt Zweil, Okechukwu Wisdom, Tarek El-Eter, Edison Lee, Michael Dong, Alan Li, Anto Joseph, Duy Nguyen, Gajesh Naik, Gautham Anant, Soubhik Deb, and Justin Drake. Efficient record-and-replay arithmetic for quantum elliptic-curve point addition. arXiv:2609.28882, 2026. Version 1, 24 September 2026.

[Por20] Thomas Pornin. Double-odd elliptic curves. Cryptology ePrint Archive, Paper 2020/1558, 2020.

[Por24] Thomas Pornin. A prime-order group with complete formulas from even-order elliptic curves. *IACR Communications in Cryptology*, 1(1), 2024.

[PZ03] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Information and Computation*, 3(4):317–344, 2003.

[RNSL17] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. In *Advances in Cryptology – ASIACRYPT 2017*, pages 241–270. Springer, 2017.

[Sch26] André Schrottenloher. Optimized point addition circuits for elliptic curve discrete logarithms. arXiv:2606.02235; IACR Cryptology ePrint Archive, Report 2026/1128, 2026.

[SFG25] Douglas Stebila, Scott Fluhrer, and Shay Gueron. Hybrid key exchange in TLS 1.3. Internet-Draft draft-ietf-tls-hybrid-design-16, 2025. Work in progress; version cited in the previous manuscript.

[Sho94] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, page 124–134, USA, 1994. IEEE Computer Society.

[SS25] Gyeongju Song and Hwajeong Seo. Quantum circuit for Curve25519 with fewer qubits. In *Information Security and Cryptology – ICISC 2024*, pages 231–243. Springer, 2025.

A Reproducibility

The implementation in `ed25519-scalar-mult` is written in Python with its own circuit representation, resource counters and simulators. It requires Python 3.12 or later (3.12.13 was used here), NumPy and `tabulate`, with `pytest` for testing. Dependencies are declared in `pyproject.toml` but are not fully version-locked. The source code is available at

https://github.com/yintong16/qsm-release.

On macOS/Linux, run:

```
./setup.sh
./paper table1  # Square-subtraction components
./paper table2  # Field-arithmetic primitives
./paper table3  # Constant-point addition
./paper table4  # Complete Wei25519 scalar schedules
./paper table5  # Odd-only encoding savings
./paper table6  # Overall comparison
```

The setup script prepares the project environment, which the wrapper selects without activation. The README documents the settings, manual installation and Windows instructions. Use `./paper -list` to list commands and `-out <new-directory>` to choose an output location. Full-width builds require several GB of memory, and full scalar runs can take a few minutes. No quantum processor, GPU or external quantum SDK is required.

CCX counts are measured directly from freshly constructed circuits. Peak logical width is the maximum number of simultaneously live qubits, including temporary workspace. Complete scalar schedules are constructed and counted with a persistent allocator, including initialization and all 28 additions, rather than extrapolated from standalone addition costs. Component sums are checked against whole-circuit counts. For Table 5, the exclusive component savings must sum to the independently measured signed-minus-odd difference, and the remaining CCX regions must match.

In Table 6, the two Schrottenloher rows are quoted literature estimates [Sch26], not remeasurements. Only the Wei25519 row is rebuilt.

Each run writes LaTeX, Markdown and JSON outputs to a new directory under `runs/benchmarks/`, preserving existing results. Records include source and configurations, timestamps, applicable seeds, resource counts, diagnostics and progress logs.

To reproduce the empirical GCD estimates, run the commands from the root of the released package. The command

```
python3 scripts/gcd_heuristics.py iterations --samples 2000
```

reports the mean μ and population standard deviation σ of the jump-2 GCD iteration count, together with the normalized coefficients μ/n and $\sigma/\sqrt{n}$.

The command

```
python3 scripts/gcd_heuristics.py width --samples 2000
```

reports the average decrease in operand bit length per iteration, obtained by fitting a straight line with a free intercept to each sampled trajectory and averaging the fitted decreases.

Both commands sample uniformly from the nonzero elements of $\mathbb{F}_q$, print their calculation method, and accept –samples to control the number of sampled inputs. Only the Python standard library is required.

Validation includes small-instance tests, value, phase and ancilla-cleanup checks, and symbolic checks over all measurement outcomes for selected inputs. These checks do not establish correctness for every input, and the tuned arithmetic remains approximate. The table commands rebuild the circuits required for each benchmark.