Quantum Discrete Logarithms on Elliptic Curves with $5n/2 + o(n)$ Logical Qubits and $\widetilde{O}(n^2)$ Toffoli Gates

Sunghyeon Jo[1] and Gye Jin Lee[1]

Georgia Institute of Technology and QED Audit
{sjo65,gyejinlee}@gatech.edu

Abstract. We give a quantum algorithm for the elliptic curve discrete logarithm problem over an n-bit prime field using $\frac{5}{2}n+o(n)$ logical qubits and $\widetilde{O}(n^2)$ Toffoli gates. Luo et al.'s exact space-efficient affine construction uses $3n+O(\log n)$ qubits with $O(n^3/\log n)$ Toffoli gates; ours lowers the leading qubit term from $3n$ to $\frac{5}{2}n$ and the Toffoli count to near-quadratic.

We construct an exact in-place modular inverter with $\frac{3}{2}n + o(n)$ logical qubits and $\widetilde{O}(n)$ Toffoli gates. For consecutive Euclidean remainders $R > r$ and coefficient magnitudes $0 \le T < t$, the invariant $Rt + rT = p$ allows us to store only three of R, r, T, t in the persistent Euclidean state and reconstruct the fourth on demand. Temporary Euclidean data are recovered from the updated state and erased after each update. The arithmetic queries used in these steps are implemented in sublinear workspace using measurement-based uncomputation. We then use Hua's identity to reduce the variable-square multiplications needed for affine point addition to inversions, yielding exact controlled point addition within $\frac{5}{2}n + o(n)$ logical qubits and $\widetilde{O}(n)$ Toffoli gates.

1 Introduction

Shor's algorithm solves discrete logarithms in finite Abelian groups in polynomial time [28]. For elliptic curves, its resource requirements depend on the arithmetic used to add points. Let p be an odd prime, $n = \lceil \log_2 p \rceil$, and $E/\mathbb{F}_p$ a nonsingular curve in short Weierstrass form. An affine point occupies two n-qubit coordinate registers and a flag for the point at infinity. The challenge is to implement the group law with little additional quantum memory while retaining fast arithmetic.

Existing constructions exhibit a time–space tradeoff. Fast classical multiplication, division, and extended GCD give $\widetilde{O}(n)$-size Boolean circuits for affine point addition [3,26,21,11], and standard reversible simulation [2] yields $\widetilde{O}(n^2)$-gate ECDLP circuits with $\widetilde{O}(n)$ qubits. Recent space-efficient constructions give explicit widths near $3n$ qubits, with near-cubic or quartic gate bounds (Table 1). We obtain both $\frac{5}{2}n+o(n)$ logical qubits and $\widetilde{O}(n^2)$ Toffoli gates, using $n/2+o(n)$ additional qubits beyond the two coordinate registers. The space reduction relies on measurement-based uncomputation: workspace is recycled by mid-circuit measurements, with the induced phases corrected before the next operation.

1.1 Main Results

Theorem 1 (Quantum Algorithm for ECDLP). *Let p be an odd n-bit prime and $E/\mathbb{F}_p$ an elliptic curve in short Weierstrass form. Given $G \in E(\mathbb{F}_p)$ and $Q \in \langle G \rangle$, there is a quantum algorithm that returns d satisfying $Q = dG$ with probability at least $2/3$, using*

$$\frac{5}{2}n + o(n) \text{ logical qubits} \qquad \text{and} \qquad \widetilde{O}(n^2) \text{ Toffoli gates.}$$

The subgroup order is recovered during classical postprocessing.

The main arithmetic result is an exact in-place modular inverter using $\frac{3}{2}n + o(n)$ logical qubits, including its n-qubit input, and $\widetilde{O}(n)$ Toffoli gates (Theorem 3). Luo et al. [20] use $2n + 6\lfloor \log_2 n \rfloor + O(1)$ qubits and $229n^2 + O(n \log n)$ Toffoli gates for their exact forward inverter, which retains its Euclidean state for later uncomputation.

Combining this inverter with Hua's identity gives the following controlled point-addition circuit.

Theorem 2 (Controlled Point Addition). *For every fixed $Q_0 \in E(\mathbb{F}_p)$, controlled point addition*

$$|c\rangle|\mathrm{enc}\, P\rangle \longmapsto |c\rangle|\mathrm{enc}(P + [c]Q_0)\rangle$$

has an exact implementation in the sense of Definition 1 on canonical point encodings (Section 2.1) using $\frac{5}{2}n + o(n)$ logical qubits and $\widetilde{O}(n)$ Toffoli gates.

For the full ECDLP algorithm, Proos and Zalka's register sharing used about $5n$ qubits [24,4]; Chevignard et al. [4] reach $3.12n + o(n)$ under heuristic assumptions with $\widetilde{O}(n^4)$ gates per run; and Luo et al. [20] give an exact affine construction using $3n + 6\lfloor \log_2 n \rfloor + O(1)$ qubits and $1056n^3/\log_2 n + O(n^2)$ gates. Thus our construction simultaneously improves the leading qubit term and the asymptotic gate bound over the space-efficient constructions in Table 1.

Table 1. Selected space-efficient ECDLP constructions over an n-bit prime field. Qubit bounds include ancillas. For Luo et al. and this work, the gate column is the total ECDLP bound. The CFS entry is a per-run bound and requires $o(n)$ runs under its stated heuristics [4, Theorem 5].

Construction	Logical qubits	Toffoli gates
Chevignard et al. [4]	$3.12n + o(n)$	$\widetilde{O}(n^4)$
Luo et al. [20]	$3n + 6\lfloor \log_2 n \rfloor + O(1)$	$1056n^3/\log_2 n + O(n^2)$
This work	$\frac{5}{2}n + o(n)$	$\widetilde{O}(n^2)$

1.2 Technical Overview

The fourth Euclidean quantity is represented implicitly by $Rt + rT = p$, so the persistent EEA state stores three of R, r, T, t. The central challenge is updating this shared representation: a Euclidean step changes all four quantities and its quotient may have n bits. We recover quotient digits, transition matrices, lengths, and layout controls from the *updated* state and erase them after each update. The four components below implement this principle within sublinear auxiliary space.

State representation. Run the extended Euclidean algorithm (EEA) on (p, z), where $z = \min(x, p-x)$ and one bit records this sign normalization. If $R > r$ are consecutive remainders and $0 \le T < t$ are the magnitudes of their alternating-sign Bézout coefficients of z, then

$$Rt + rT = p. \tag{1}$$

The identity gives the implicit value $R = (p-rT)/t$ or $T = (p-Rt)/r$, depending on the layout. Before t reaches $H_{\mathrm{sw}} = 2^{\lceil n/2 \rceil}$ we store $(r, T\,;\, t)$, and afterwards $(R, t\,;\, r)$, where the semicolon separates the main and short registers. The main pair has combined unrounded bit length $n + O(1)$; the short value is below H_{sw} before the switch and below $p/t \le 2^{n-\lceil n/2 \rceil}$ afterwards. Thus the EEA state occupies $3n/2 + o(n)$ qubits.

Arithmetic evaluation. Section 3 evaluates a bit window or predicate of a constant-depth arithmetic expression on demand using near-linear work and sublinear working memory. Products are scanned by columns; division uses a Newton recurrence with comparison-based correction, recomputing smaller-precision values as needed. The coarse scale $w = \Theta(n/\log\log n)$ is the window width and keeps the recomputation overhead polylogarithmic. A specialization of spooky pebbling [7,17] realizes these classical evaluations as arithmetic oracles using $o(n)$ ancilla qubits (Section 2.4).

Euclidean updates. A finer scale $b = \left\lceil n/\log^K(n+2) \right\rceil$ controls the EEA schedule. Within each coefficient interval, small quotients are collected into an $O(b)$-bit transition matrix determined from leading words together with boundary tests on the full EEA state; the updated coefficients recover the matrix for erasure. Large quotients are applied one b-bit digit at a time, with each digit and temporary control bit likewise recovered from the updated state. The scale w controls the recomputation cost, while b keeps the transient continued-fraction data sublinear.

Point addition. Point addition combines the inverter with a variable-square multiplication. Hua's identity [12] realizes $(X, Y) \mapsto (X, X^2Y)$ for $X \neq 0$ by five sequential inversions, additions, and a small correction. Two such permutations and affine coordinate operations implement addition by a classically known non-two-torsion point outside a constant-size exceptional set. A final permutation supported on at most eight strings handles that set, including infinity. Each inversion preserves the other coordinate and reuses the same workspace.

1.3 Related Work

Proos and Zalka [24] introduced EEA register sharing for prime-field ECDLP; Kaye and Zalka [13] studied related arithmetic over binary fields. Roetteler et al. [25] and Häner et al. [9] gave reversible binary-GCD circuits and detailed resource estimates. Luo et al. [20] refine register sharing using length registers and location-controlled arithmetic. The Proos–Zalka and Luo constructions keep all four Euclidean quantities represented; our persistent EEA state stores three.

Khattar et al. [14] distinguish explicit and implicit access to Bézout coefficients and introduce a compact operation record called a dialog; Schrottenloher [27] uses a related record-and-replay approach, with a different encoding, for prime-field point addition. Our construction recovers transient quotient data from the updated EEA state. Chevignard, Fouque, and Schrottenloher [4] use projective arithmetic in a residue number system and Legendre-symbol output compression.

Concrete ECDLP estimates appear in [18,1,27,10,19], with depth trade-offs in [15]. Our analysis is asymptotic; concrete estimates at cryptographic sizes would require optimizing the polylogarithmic factors of Sections 3 and 4, and are left open. Our phase-estimation reduction follows [16,22,8]; algorithms without prior knowledge of the group order are studied by Ekerå [6].

Organization. Section 2 fixes the encodings, EEA conventions, and circuit model. Sections 3 and 4 give the evaluator, oracle construction, and inverter; Sections 5 and 6 give affine point addition and ECDLP. The appendices give supporting details for arithmetic, in-place updates, selector correctness, simulation, phase estimation, and postprocessing.

2 Preliminaries

2.1 Elliptic Curves and Point Encodings

Throughout, p is an odd prime, $n = \lceil \log_2 p \rceil$, and

$$E/\mathbb{F}_p : y^2 = x^3 + a_E x + b_E, \qquad 4a_E^3 + 27b_E^2 \neq 0,$$

is a nonsingular curve in short Weierstrass form. The group law is written additively, with identity $\mathcal{O}$. A field element is *canonical* if it is represented by its residue in $[0, p)$ as an n-bit string. An affine point $P = (x, y)$ has encoding $\operatorname{enc} P = (x, y, 0)$ with two canonical coordinates and one flag qubit, and $\operatorname{enc} \mathcal{O} = (0, 0, 1)$; the *canonical point encodings* are the strings of this form with $P \in E(\mathbb{F}_p)$. Intermediate coordinate pairs may be off-curve.

For affine points $Q_0 = (u, v)$ and $P = (x, y)$ with $x \neq u$, the addition formulas are

$$\lambda = \frac{y - v}{x - u}, \qquad x_+ = \lambda^2 - x - u, \qquad y_+ = \lambda(x - x_+) - y. \tag{2}$$

The cases $P = \mathcal{O}$ and $x = u$, including doubling and inverse points, are handled separately in Section 5.

2.2 Extended Euclidean Algorithm

For $0 < z < p$, the Euclidean algorithm on (p, z) produces remainders $r_0 = p$, $r_1 = z$, and $r_{i+1} = r_{i-1} - q_i r_i$ with $q_i = \lfloor r_{i-1}/r_i \rfloor$, until some remainder vanishes; since p is prime, the last nonzero remainder is 1. Alongside, define the *coefficient magnitudes* $t_0 = 0$, $t_1 = 1$, $t_{i+1} = t_{i-1} + q_i t_i$. These are the absolute values of the Bézout coefficients of z, whose signs alternate.

Lemma 1. *For every $i \geq 1$ up to and including the first zero remainder,*

$$r_{i-1} t_i + r_i t_{i-1} = p \tag{3}$$

and

$$r_i \equiv (-1)^{i-1} t_i z \pmod{p}. \tag{4}$$

Proof. The first identity holds at $i = 1$, and

$$r_i t_{i+1} + r_{i+1} t_i = r_i (t_{i-1} + q_i t_i) + (r_{i-1} - q_i r_i) t_i = r_i t_{i-1} + r_{i-1} t_i.$$

For the sign identity, $r_0 \equiv 0 = t_0 z$ and $r_1 = z = t_1 z$ give the initial congruences. Substitution in the remainder recurrence propagates the alternating signs, proving (4).

After j divisions, Section 4 writes $R = r_j$, $r = r_{j+1}$, $T = t_j$, and $t = t_{j+1}$. Thus $Rt + rT = p$ and $r \equiv (-1)^j tz$. Identity (3) underlies register sharing [24]: each positive product Rt or rT is at most $p < 2^n$, so the combined bit length of its factors is at most $n + 1$. When $T = 0$, it uses no cells. Section 4.1 uses this bound to store three quantities and reconstruct the fourth.

2.3 Arithmetic Scales and Register Layout

All logarithms are binary. Write $[P]$ for the indicator of a predicate P, and $\mathrm{len}(v) = \lfloor \log_2 v \rfloor + 1$ for $v \geq 1$, with $\mathrm{len}(0) = 0$. The notation $\widetilde{O}(f(n))$ suppresses factors $\log^{O(1)}(n + 2)$; the hidden constants are independent of the operand values. The modulus, curve coefficients, and addends used by controlled point addition are classically known. Loop bounds, interval endpoints, and precision parameters depend only on the input length and classically known parameters. The parameters w and b below set two arithmetic scales. Operands that fit in working memory at the relevant scale are materialized and evaluated directly; larger expressions are evaluated through bit windows (Section 3).

We first specify the continued-fraction routine on materialized operands used later to select blocks of Euclidean quotients. For $q \geq 1$ write $M_q = \left(\begin{smallmatrix} q & 1 \\ 1 & 0 \end{smallmatrix}\right)$ for the transition matrix of one Euclidean step. Use a uniform fast continued-fraction algorithm on $O(m)$-bit inputs that returns the quotient sequence [26,21]. Its quotients have total binary length $O(m)$. Store a balanced tree of products of consecutive transition matrices; prefix matrices are reconstructed from its tree blocks. At each tree level, the total matrix bit length is $O(m)$: the bit length of a

product is bounded by the sum of $\log_2(q+1)$ over its interval, up to a constant factor. The tree therefore occupies $O(m\log(m+2))$ bits and is constructible with $m\log^{O(1)}(m+2)$ work.

A prefix matrix is obtained by multiplying the $O(\log(m+2))$ tree blocks covering its prefix. Boolean simulation [23], parsing, and these queries fit an oblivious circuit of size $O(m\log^{c_{\mathrm{CF}}}(m+2))$ for some constant $c_{\mathrm{CF}} \geq 1$. On materialized inputs, the intermediate wire values for one prefix query are retained as working memory, which is reused by subsequent queries.

The two scales are needed for different resource constraints. With the choices below, using w for the continued-fraction routine would leave $O(w\log^{c_{\mathrm{CF}}}(n+2))$ retained wires, which is not $o(n)$; using b as the window width would give the query-tree bound $\exp(O((\log\log n)^2))$, which is superpolylogarithmic. Choose a fixed integer $K > c_{\mathrm{CF}}$ and set

$$\begin{aligned} &g = \max\{4, \lceil \log_2 \log_2(n+16)\rceil\}, \quad w = \lceil n/g \rceil, \\ &b = \left\lceil n/\log_2^K(n+2) \right\rceil, \qquad\qquad h = \lceil n/2 \rceil, \qquad H_{\mathrm{sw}} = 2^h, \quad B = 2^b. \end{aligned} \tag{5}$$

Here w is the window width of Section 3, while b bounds the transition matrices and retained continued-fraction circuits in Section 4. Define

$$s_*(n) = w(1+\log g) + b\log^{c_{\mathrm{CF}}}(n+2) + \log^2(n+2). \tag{6}$$

With these choices,

$$g^{O(1+\log g)} = \log^{o(1)}(n+2), \qquad w(1+\log g) = o(n), \qquad b\log^{c_{\mathrm{CF}}}(n+2) = o(n),$$

while $n/w = O(g)$ and $n/b \leq \log^K(n+2)$. Hence $s_*(n) = o(n)$. Lemma 3 uses the first two bounds for the evaluator, and the third bounds the retained continued-fraction circuit at scale b; the bound on n/b keeps the interval and digit schedules polylogarithmic.

These scales also determine the register allocation. The inverter uses a *main register* and a *short register* of

$$N_{\mathrm{main}} = n + 8b + 16, \qquad N_{\mathrm{short}} = h + 8b + 16 \tag{7}$$

qubits. A *subregister* is a contiguous block of cells of one register that holds one integer; an integer v is assigned the rounded length $\mathrm{cap}_b(v) = b\lceil \mathrm{len}(v)/b \rceil$. The main register holds two subregisters anchored at its two ends, the right one in reverse bit order, so that both grow towards the middle; the short register holds one subregister anchored at its left end. *Length registers* of $O(\log n)$ qubits store the rounded lengths of the three stored integers, and one layout bit records which of the two layouts of Section 4.1 is active; meta denotes these quantum length and layout registers collectively. Unused cells contain zero. During an update, the subregister being changed is the *target*; the cells it may occupy, namely the larger of its old and new rounded lengths, form the target region. Every other subregister is *preserved*, and the target region must be disjoint from the preserved subregisters. We prove this disjointness for each operation, including its intermediate states. All asymptotic arguments concern sufficiently large n; the remaining finitely many sizes are handled by hard-wired circuits.

2.4 Circuit Model and Cost

Following [4,20], we count logical qubits and Toffoli gates. The qubit count is the maximum number of qubits in use at any time, including data, control, and ancilla registers; ancillas start and end in $|0\rangle$. Classical working memory and measurement records are accounted for separately from the logical-qubit count and remain polynomially bounded. The reversible circuits below use NOT (X), CNOT, and Toffoli gates; uncomputation additionally uses Hadamard gates, computational-basis measurements, resets, and classically controlled Pauli Z gates. Each Toffoli bound stated below also bounds the number of these other operations up to a constant factor. The ECDLP algorithm alone uses one-qubit phase rotations, accounted for in Section 6.

An *adaptive circuit* permits later gates to depend on earlier measurement outcomes. All measurement positions in the constructions below are chosen in advance. A *Boolean program* uses bounded-fan-in AND, XOR, NOT, and bit assignments on a mutable working array, with read access to preserved input. We call such a program *oblivious* if its instruction sequence and accessed bit positions depend only on the input length and classically known parameters, not on the input values or measurement outcomes; data-dependent cases are executed under controls within this input-independent schedule. Circuit generation and classical processing take polynomial time and space. The cost model assumes arbitrary logical connectivity. In diagrams, a slash marks a register, a dot a control, a filled triangle an updated output, and a double wire a classical record.

We write $X \leftarrow e$ for an in-place replacement: the subregister currently holding X is overwritten with the value e. The right-hand side is evaluated before the overwrite. A displayed sequence of replacements is executed in order, so a value produced on one line is available on later lines, while symbols not yet replaced retain their current values. We write $\longmapsto$ for the resulting map on values or on encoded states, as in (16), and $(u, v\,;\, s)$ for the register layout of the main and short registers. Primes denote the state after a block of Euclidean steps; the subscripts old and new refer to a single controlled update.

Measurement-based uncomputation [7,17] recycles a register containing a Boolean function of preserved data. On a Hadamard-basis outcome $y \in \{0,1\}^\ell$, its unnormalized branch map is

$$|X\rangle|f(X)\rangle \longmapsto 2^{-\ell/2}(-1)^{y\cdot f(X)}|X\rangle|0\rangle. \tag{8}$$

Recomputing $f(X)$ and applying the corresponding Pauli Z operations cancels the phase. The following definition records the exactness required when such steps are composed.

Definition 1 (Exact Implementation). *Let U be a permutation unitary on a specified subspace of data encodings. For an adaptive circuit with initially zero workspace and a complete measurement record μ, let K_μ be the Kraus operator for outcome record μ, including the classically controlled corrections and resets. The circuit implements U exactly if*

$$K_\mu\big(|\psi\rangle|0\rangle\big) = \alpha_\mu U|\psi\rangle|0\rangle, \qquad \sum_\mu |\alpha_\mu|^2 = 1, \tag{9}$$

for every $|\psi\rangle$ in the specified data subspace, where α_μ is independent of $|\psi\rangle$. For a controlled operation the control qubit is part of the data.

Because α_μ does not depend on $|\psi\rangle$, the definition covers inputs entangled with a reference system. It is preserved by composition: the record of a composition is the concatenation of the records, and the branch scalars multiply.

An *arithmetic oracle* is an exact implementation of the XOR map

$$O_f : |X\rangle|z\rangle \longmapsto |X\rangle|z \oplus f(X)\rangle$$

for a bit-string-valued function f of the preserved input X, with all phase corrections completed and all workspace returned to zero before the next operation. Oracle calls are implemented subcircuits, and their gates and workspace are included in every resource bound. The input is preserved only for the duration of the call; a later update may change it, and a later oracle may take the changed value as its input. The inverse construction reverses the logical updates and makes fresh oracle calls for the values it needs; measurement outcomes remain classical records.

3 Arithmetic Evaluation

The inverter reserves $o(n)$ ancilla qubits for temporary computation, while omitted values, products, and quotients can have $\Theta(n)$ bits. We represent such values implicitly by constant-depth arithmetic expressions and evaluate the bit windows and predicates needed by the inverter. We give a classical window evaluator, its quantum implementation, and the resulting in-place updates; Section 4 uses these three interfaces.

3.1 Expressions and Bit-Window Queries

An *arithmetic expression* is a constant-size, constant-depth formula whose leaves are preserved subregisters, preserved temporary buffers, or classically known $O(n)$-bit constants such as p. Its internal nodes are signed addition and subtraction, multiplication, floor division and remainder, shifts, and inversion of an odd integer modulo a power of two; intermediate values have $O(n)$ bits. Size and depth refer to arithmetic-operation nodes. We also allow sign, equality and order tests, bit length, and the 2-adic valuation of a nonzero value. Undefined cases return a designated default: a domain test substitutes valid dummy operands and masks the answer. A stack frame stores an $O(\log n)$-bit description of the current node and its parameters, not the represented integer. For a nonnegative expression e, offset s, and $0 \le \ell \le w$, define

$$\mathsf{Window}(e, s, \ell) = \lfloor e/2^s \rfloor \bmod 2^\ell.$$

A *bit-window query* $\mathsf{Query}(e, s, \ell)$ returns this window into a buffer supplied by the caller, which may be reused once the answer has been consumed. A *predicate*

query returns one bit; bit-length and valuation queries return an $O(\log n)$-bit integer. Signed values use sign and magnitude. Throughout, queries read the preserved subregisters, length registers, and layout bit in place.

Values materialized in working memory are processed with standard fast algorithms.

Lemma 2 (Arithmetic on Materialized Operands). *Multiplication, division by a nonzero integer, and inversion of an odd integer modulo a power of two on materialized $O(m)$-bit operands have oblivious Boolean programs using $m \log^{O(1)}(m+2)$ operations and $O(m)$ working bits.*

The proof (Appendix A) is the usual reduction to fixed-point convolution [3] with sufficient precision to recover the integer output exactly.

The evaluator works digitwise in radix 2^w, in which an $O(n)$-bit integer has $O(g)$ digits. A digit of a sum is obtained by scanning the lower columns and retaining one carry. A digit of a product is obtained by a column scan over the lower digit products (Algorithm 4, Appendix A): each child query returns one digit into a buffer, the digit product is accumulated, and the buffer is then reused. A window that crosses a digit boundary is assembled from two aligned digits. Comparisons and zero tests retain $O(1)$ result bits during a complete digit scan; lengths and scan indices require $O(\log n)$ bits; a variable shift reads two adjacent windows and shifts them within the buffer.

Division is reduced to the reciprocal

$$R_m(D) = \left\lfloor 2^{\ell_D+m}/D \right\rfloor, \qquad \ell_D = \lfloor \log_2 D \rfloor,$$

at precision m, and the reciprocal to itself at half precision. For $m > w$ put $m' = \lceil m/2 \rceil$ and define the expressions

$$y = 2^{m-m'} R_{m'}(D), \qquad \tau = 2y - \left\lceil D y^2 / 2^{\ell_D+m} \right\rceil.$$

Then $R_m(D) = \tau + [D(\tau+1) \le 2^{\ell_D+m}] + [D(\tau+2) \le 2^{\ell_D+m}]$ (Appendix A.3): the integer τ is at most two below $\left\lfloor 2^{\ell_D+m}/D \right\rfloor$, so two comparisons recover the correct integer. The evaluator recomputes digits of $R_{m'}(D)$ whenever a digit of τ needs them. At $m \le w$ the leading $m+2$ bits of the divisor give a base-case candidate that is at most one too large, and one comparison against the full divisor corrects it. Algorithm 1 makes the calls explicit. Inversion modulo 2^m of an odd D uses the same halving schedule through $V_m = V_{m'}(2 - DV_{m'}) \bmod 2^m$ (Appendix A.4). A quotient $\lfloor A/D \rfloor$ with $A < 2^N$ is $\left\lfloor A\, R_m(D)/2^{\ell_D+m} \right\rfloor$ at $m = N+2$, corrected upward by one comparison (Appendix A.3); a remainder is $A - D\lfloor A/D \rfloor$.

We next bound the resulting query tree. One product digit makes $O(g^2)$ queries for operand windows by the two loops of Algorithm 4. Each reciprocal step above precision w expands into a constant number of additions, products, shifts, and comparisons involving D and $R_{\lceil m/2 \rceil}(D)$. Rounding upward is implemented by a shift and a zero test on the discarded low bits. Since each scan has polynomially many child queries in g, this expansion has at most $C_{\mathrm{q}} g^{c_{\mathrm{q}}}$ child

Algorithm 1 A window of the reciprocal $R_m(D)$.

Input: Positive divisor represented by an expression D, precision m, window (s, ℓ) with $\ell \le w$.

$\ell_D \leftarrow \mathrm{len}(D) - 1$ by a digit scan.
if $m \le w$ **then**
  $e_D \leftarrow \max\{0, \ell_D + 1 - (m+2)\}$; store $D_0 = \lfloor D/2^{e_D} \rfloor$ in at most $m+2$ bits.
  $q_0 \leftarrow \lfloor 2^{\ell_D + m - e_D} / D_0 \rfloor$ using Lemma 2.
  $z \leftarrow [\, Dq_0 > 2^{\ell_D + m} \,]$ by a scan over the whole divisor.
  return $\mathsf{Window}(q_0 - z, s, \ell)$.
else
  $m' \leftarrow \lceil m/2 \rceil$; define expressions $y = 2^{m-m'} R_{m'}(D)$, $\tau = 2y - \lceil Dy^2 / 2^{\ell_D + m} \rceil$.
  $z_1 \leftarrow [\, D(\tau+1) \le 2^{\ell_D+m} \,]$, $z_2 \leftarrow [\, D(\tau+2) \le 2^{\ell_D+m} \,]$ by digit queries.
  return $\mathsf{Query}(\tau + z_1 + z_2, s, \ell)$.

An expression records its arithmetic syntax; evaluation occurs through queries. The corrections query the *uncorrected* τ, and the branch depends only on the precision m.

queries for constants $C_{\mathrm{q}}, c_{\mathrm{q}}$ independent of m. The same reasoning applies to quotient correction and odd inversion. The only recursive calls are the precision-halving reciprocal and odd-inverse calls; at $m \le w$, reciprocal evaluation retains one $O(w)$-bit base-case candidate and does not recurse.

Lemma 3 (Arithmetic Evaluation). *For an expression of constant depth d with preserved $O(n)$-bit inputs, one window of at most w bits, one predicate, or one bit-length or valuation query has an oblivious Boolean evaluator using*

$$\begin{aligned} \mathcal{T} &= n \log^{O(1)}(n+2) && \text{operations}, \\ \mathcal{S} &= O\big(w(1+\log g) + \log^2(n+2)\big) && \text{working bits}. \end{aligned} \tag{10}$$

The bounds include input access. Constants may depend on the formula and its intermediate bit-length bound, but not on the operand values.

Proof. Recursion. Signed operations reduce to the nonnegative digit scans above, with sign tests and, for floor division, an additional remainder test. A reciprocal correction calls only the uncorrected Newton expression and smaller-precision reciprocals. A base-case reciprocal correction retains its $O(w)$-bit candidate and scans the divisor expression; it never recurses on the candidate being corrected. Thus all calls are well founded.

Expand each composite operation into its scans and each division or inverse into its precision-halving recursion. Along any root-to-leaf path the formula depth decreases at most d times and each division or inverse contributes $O(1 + \log g)$ precision levels, so the resulting depth is $D_{\mathrm{eval}} = O_d(1 + \log g)$. Let N_j bound the number of query nodes below j remaining levels. For constants $C_{\mathrm{q}}, c_{\mathrm{q}}$,

$$N_0 = 1, \qquad N_{j+1} \le 1 + C_{\mathrm{q}} g^{c_{\mathrm{q}}} N_j.$$

Consequently the whole query tree has

$$N_{D_{\mathrm{eval}}} \le g^{O_d(1+\log g)} = \exp\big(O_d((\log\log\log(n+16))^2)\big) = \log^{o(1)}(n+2) \quad (11)$$

nodes. Constants and temporary buffers are terminating leaves of this tree.

Work. Reading a stored operand scans all $O(g)$ aligned slots, copies the selected slot and its neighbour under address controls, masks cells outside the subregister, and shifts within the buffer. It costs $O(n + w\log n)$ Boolean operations and $O(w)$ working bits. Reversed subregisters use reversed indices. A basic internal product-scan node performs $O(g^2)$ operations on materialized digits; other constant-size expansions contribute at most $g^{O(1)}$ such work. By Lemma 2, their cost per node is bounded by $g^{O(1)}w\log^{O(1)}(w+2)$. Charging the larger of this bound and the register-access bound to every node of (11) gives $\mathcal{T} = n\log^{O(1)}(n+2)$.

Space. Depth-first evaluation retains only its active stack. Each frame contains an $O(\log n)$-bit expression description and counters, a constant number of returned windows, and scan state of $O(w + \log g)$ bits. In particular, a parent may retain an operand window as well as its accumulator while a child executes. Sibling calls reuse buffers; a call on materialized operands uses $O(w)$ additional bits. The $O_d(1+\log g)$ active frames use the stated $\mathcal{S}$ bits, including all retained parent data.

Obliviousness. Loop and recursion bounds depend only on the input length. Data-dependent offsets are implemented by controlled scans over the stored operand. Each stack depth has its own working region, and the formula determines the possible frame types. Executing data-dependent cases under controls therefore gives an oblivious schedule within the same bounds.

3.2 Measurement-Based Uncomputation

We use measurement-based uncomputation in the spooky pebbling framework of Gidney and Kornerup–Sadun–Soloveichik [7,17]. We apply the three-pebble line strategy to the evaluator, leaving the preserved input outside the checkpoints and recomputing only the working configuration.

Lemma 4 (Simulation). *Consider an oblivious Boolean program with preserved input X. It uses $\mathcal{T} \ge \mathcal{S}$ operations and $\mathcal{S}$ working bits, and returns $O(\mathcal{S})$ output bits. Its XOR map has an exact implementation using $O(\mathcal{S})$ ancilla qubits and $O(\mathcal{T}^2/\mathcal{S} + \mathcal{T})$ Toffoli gates. For every measurement record, the resulting scalar is independent of all input bits, including an external control.*

Proof. Divide the program into $Q_{\mathrm{seg}} = \lceil \mathcal{T}/\mathcal{S} \rceil$ segments. Each segment has an XOR circuit mapping one configuration to the next with $O(\mathcal{S})$ gates and ancillas, obtained by recording and reversing only that segment. The preserved input X remains in place outside the checkpoints. Apply the three-pebble line strategy [17, Corollary 3.4] to these segment circuits. Its cost is $O(Q_{\mathrm{seg}}^2\mathcal{S})$ with $O(\mathcal{S})$ live workspace. Each discarded checkpoint contributes the character phase in (8); ghost removal cancels it, leaving only an input-independent normalization. Appendix D gives the schedule and branch invariant explicitly.

Corollary 1. *Every query of Lemma 3 can be realized as an arithmetic oracle with $\widetilde{O}(n)$ Toffoli gates and $O(s_*(n)) = o(n)$ ancilla qubits. The same holds when the evaluator additionally retains the $O(b \log^{c_{\mathrm{CF}}}(n+2))$ wires of a continued-fraction circuit at scale b.*

Proof. Pad the working memory to $\mathcal{S} = \Theta(s_*(n)) \geq w$. Since $n/w = O(g)$, $\mathcal{T}^2/\mathcal{S} \leq n \log^{O(1)}(n+2)$. The retained wires add sublinear memory and near-linear work. Apply Lemma 4.

3.3 In-Place Updates

The oracles supply digits of an implicitly represented integer while a disjoint target changes. The following lemma, proved in Appendix B, turns them into permutations of the target. The third item is the one Section 4 uses most: it replaces a stored Euclidean quantity by another one that is linearly related to it through the preserved quantities.

Lemma 5 (In-Place Updates). *For an $m = O(n)$-bit target, each of the following has an exact implementation with $\widetilde{O}(n)$ Toffoli gates and $O(s_*(n))$ ancilla qubits:*

1. *addition modulo 2^m of an expression in preserved inputs;*
2. *multiplication modulo 2^m by a preserved odd expression, and its inverse;*
3. *replacement of x by y under an integer relation*

$$\mathsf{A}x + \mathsf{B}y = \mathsf{C}, \qquad 0 \leq x, y < 2^m, \tag{12}$$

where $\mathsf{A}, \mathsf{B}, \mathsf{C}$ are expressions in preserved inputs and $\mathsf{A}, \mathsf{B} \neq 0$.

The third map is a permutation of all m-bit target strings that acts as $x \mapsto y$ on the strings satisfying (12)*; after removing common powers of two, the nontrivial case of an even coefficient uses a cyclic rotation within the same target. The proof also handles valuations at least m, without extending the target.*

A replacement may change the target's length. Compute the old and new rounded lengths $\mathrm{cap}_b(x), \mathrm{cap}_b(y)$ into temporary length registers, take their maximum as the target region, which must be disjoint from the preserved subregisters, and perform the replacement there. Afterwards shrink to the new length, and recover the old value from the expression

$$x = (\mathsf{C} - \mathsf{B}y)/\mathsf{A}, \tag{13}$$

which determines $\mathrm{cap}_b(x)$ and clears the temporary registers while $\mathsf{A}, \mathsf{B}, \mathsf{C}$ are still preserved. For an addition, the inverse expression is $x = y - a$; a controlled operation on its inactive branch uses its unchanged lengths. Lengths range over $O(n/b+1)$ input-independent values. Before a controlled scan over these values, save the $O(\log n)$-bit temporary registers that select the length and layout case and preserve them during the scan; afterwards update meta and uncompute these temporary registers from the updated state using the formulas in Section 4. The moves and exchanges of subregisters are predetermined permutations of cells, applied under these controls, and cost $O(n)$ gates per case, hence $n \operatorname{polylog} n$ in total.

4 Space-Efficient Modular Inversion

We now run the EEA of Section 2.2 in the two registers of (7). The persistent EEA state stores three of R, r, T, t; (3) supplies the fourth on demand. We partition the growth of the coefficient t into input-independent intervals, batch the small quotients of an interval into one transition matrix, process a large quotient by digits, and design every update so that its temporary data can be recomputed from the updated state. Throughout this section, "oracle" means an oracle of Corollary 1, and every replacement is an instance of Lemma 5.

4.1 Euclidean State

For $0 < x < p$, compute $\eta_0 = [x > p/2]$ and normalize the sign by conditionally replacing x with $p - x$:

$$|x\rangle|0\rangle \longmapsto |x\rangle|\eta_0\rangle \longmapsto |z\rangle|\eta_0\rangle, \qquad z = \min(x, p - x).$$

Keeping η_0 makes this normalization reversible, and $z \le p/2$ makes the first quotient at least 2. Initialize

$$(R, r, T, t) = (p, z, 0, 1). \tag{14}$$

The register content is $(r, T\,;\, t) = (z, 0\,;\, 1)$: the original target is the left main subregister and now holds r, the right main subregister holds $T = 0$, and the short register holds $t = 1$; $R = p$ is supplied as a classically known constant. Write the rounded lengths $\mathrm{cap}_b(z), 0, \mathrm{cap}_b(1)$ into the length registers and set the layout bit to the early layout. The same sign qubit now holds η, initially η_0, and is toggled once per division, so that after j divisions $\eta = \eta_0 \oplus (j \bmod 2)$.

At interval boundaries the EEA state satisfies

$$\begin{gathered} Rt + rT = p, \qquad R > r > 0, \qquad 0 \le T < t, \\ T = 0 \quad \Longrightarrow \quad R \ge 2r. \end{gathered} \tag{15}$$

A division with $q = \lfloor R/r \rfloor$ maps

$$(R, r, T, t) \longmapsto (r,\ R - qr,\ t,\ T + qt), \tag{16}$$

and we stop at $r = 1$. Lemma 1 supplies the identity. Since $Rt + rT = p$ implies $\gcd(R, r) \mid p$ and $0 < r < p$, primality gives $\gcd(R, r) = 1$; hence $R - qr > 0$ when $r > 1$; the new coefficient $T + qt$ exceeds t, since $T > 0$ or, by the last line, $q \ge 2$; and the last line survives because the new smaller coefficient is positive. The last line is maintained as an invariant and makes the reverse coefficient divisions of Section 4.3 agree with the ordinary continued-fraction expansion.

The state is stored in one of two layouts:

$$\begin{cases} (r, T\,;\, t), & t < H_{\mathrm{sw}}, \qquad \text{implicit } R = (p - rT)/t, \\ (R, t\,;\, r), & t \ge H_{\mathrm{sw}}, \qquad \text{implicit } T = (p - Rt)/r. \end{cases} \tag{17}$$

For positive u, v with $uv < 2^n$, $\mathrm{len}(u) + \mathrm{len}(v) \le n + 1$; hence by (15) the two main values have combined bit length at most $n + 1$. Rounding their capacities to multiples of b uses $n + O(b)$ cells in either layout; a zero coefficient uses no cells. In the late layout $r < p/t \le 2^{n-h} \le H_{\mathrm{sw}}$, so r fits the short register. Both layouts fit (7), and because t only increases, the layout changes at most once. Figure 1 shows the two layouts. The switch replaces a growing short coefficient by a shrinking short remainder.

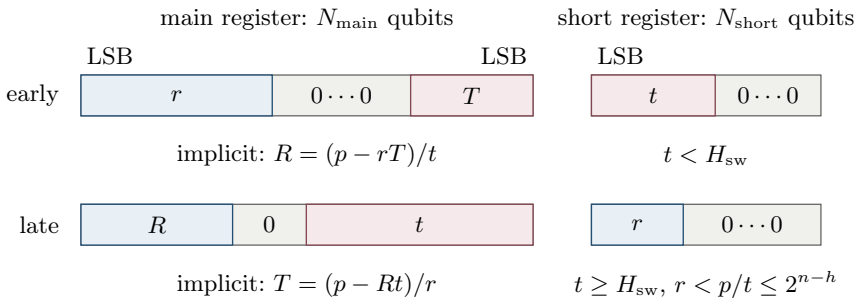

Fig. 1. Early and late layouts. Blue fields are remainders, red fields are coefficients, and gray cells are zero. The right main subregister is reversed so the two main fields grow toward each other. Widths are rounded to multiples of b; workspace and length and layout data are omitted.

4.2 Coefficient Intervals

Which quotients occur, and how large they are, depends on x, so the quantum circuit uses an input-independent schedule based on the increasing coefficient t. For $i = 0, \ldots, \lceil n/b \rceil$ let

$$[L, U) = [2^{ib}, 2^{(i+1)b}). \tag{18}$$

A quotient is *small* if $q < B$ and *large* otherwise. In each interval, first process consecutive small quotients until $r = 1$, $t \ge U$, or the next quotient is large: compute their transition matrix, apply it, and recover it from the output for erasure. Then perform one division with a large quotient if $r > 1$, $t < U$, and $q \ge B$. The circuit executes the same $\lceil n/b \rceil$ digit positions in every interval and acts as the identity when this division is inactive.

Lemma 6 (Invariant at an Interval Boundary). *At entry to every interval the state satisfies* (15) *and*

$$T < L \le t, \qquad \text{or} \qquad r = 1 \text{ and } t < L. \tag{19}$$

At exit the same holds with U in place of L. At most one division with a large quotient occurs per interval, and after the last interval $r = 1$.

Proof. Initially $T = 0 < 1 = L \le t$. If the small quotients reach U, the first crossing has $T = t_{\text{old}} < U$. If a large quotient occurs first, then $L \le t < U$ and $T + qt \ge Bt \ge BL = U$, so that division gives the first crossing of U. Otherwise $r = 1$ below U. A state already beyond U has $T < L < U$ and is unchanged. Equation (16) preserves (15); since $t < p < 2^n$, a nonterminal state cannot satisfy the exit condition of the last interval, whose upper endpoint exceeds 2^n.

There are $O(n/b+1)$ intervals and $O((n/b+1)^2)$ digit positions in the schedule. Every oracle clears its ancillas and phase corrections before the next data update.

Each interval consists of the small-quotient phase (Algorithm 2) followed by the optional large-quotient phase (Algorithm 3).

4.3 Small Quotients

With M_q as in Section 2.3, a block of small quotients $q_1, \dots, q_k$ has transition matrix

$$M = M_{q_1} \cdots M_{q_k} = \begin{pmatrix} a & b_1 \\ c & d \end{pmatrix}, \qquad \varepsilon = \det M \in \{-1, 1\}; \tag{20}$$

an empty block has $M = I$. Primes denote the state after the block. Since each M_q is symmetric, iterating (16) gives

$$\begin{pmatrix} R \\ r \end{pmatrix} = M \begin{pmatrix} R' \\ r' \end{pmatrix}, \qquad t' = at + cT, \quad T' = b_1 t + dT, \quad rt' = cp + \varepsilon r' t. \tag{21}$$

For a nonempty block, $c > 0$ and every entry is at most a. At entry, (19) gives $L \le t$, while nonemptiness implies $t < U$; during the block, t remains below U before its final division. Thus

$$t' < (B+1)U, \qquad a \le t'/t < (B+1)B < 2^{2b+1} : \tag{22}$$

M has $O(b)$ bits. It can be both computed from the input and recomputed from the output.

Lemma 7 (Selection and Recovery). *For an interval input satisfying* (15) *and* (19)*, stored as in* (17)*, there are oblivious Boolean evaluators that (i) compute the transition matrix M of the maximal block of small quotients from the input and (ii) recover M from the block output and the known endpoint L. Each uses $n \log^{O(1)}(n+2)$ operations and $O(s_*(n))$ working bits.*

Proof (Sketch). For selection, first test the stopping conditions $r = 1$, $t \ge U$, and whether the next quotient is large, $R \ge Br$. Otherwise the next quotient is small. If $\operatorname{len}(R) \le \ell_{\text{CF}}$, where $\ell_{\text{CF}} = O(b)$ is the materialization threshold defined in Appendix C, materialize (R, r) in working memory and run the continued-fraction circuit on that pair. Otherwise, sufficiently many leading bits of R and r determine a common continued-fraction prefix, as in Lehmer's GCD algorithm [3]. The candidate list is truncated before a quotient that would produce remainder

zero or is at least B, and tests on the full EEA state associated with each prefix matrix locate the first occurrence of $r = 1$ or $t \geq U$.

If the two leading-word endpoint expansions disagree, the actual ratio R/r is close to a continued-fraction boundary. The separation argument of Appendix C shows that at most two additional Euclidean divisions on the full EEA state are needed to determine the block. All continued-fraction data at scale b fit within the $O(b \log^{c_{\mathrm{CF}}}(n+2))$ retained-wire term of $s_*(n)$, and the boundary tests are constant-depth expressions handled by Lemma 3.

For recovery, T' is the value of t immediately before the last division of a nonempty block, so $T' \geq L$; an empty block instead has $T' < L$. Ordinary Euclid on (t', T') recovers the quotients in reverse order, using the indexing of Section 2.2:

$$t_{i+1} = q_i t_i + t_{i-1}, \qquad 0 \leq t_{i-1} < t_i. \tag{23}$$

Stop when the smaller coefficient drops below L. This yields M^{T}, hence M; strict coefficient growth determines each reverse quotient uniquely. Appendix C gives the endpoint conventions, boundary argument, and full resource analysis.

Corollary 1 turns both directions into arithmetic oracles with $\widetilde{O}(n)$ Toffoli gates and $o(n)$ ancillas; the transition matrix has $O(b)$ bits, and the continued-fraction circuit at scale b occupies $O(b \log^{c_{\mathrm{CF}}}(n+2))$ bits. The block data are erased before the next phase, keeping live continued-fraction storage sublinear.

Algorithm 2 gives the complete small-quotient phase, including selection and cleanup. For a nonempty block, write $M = \left(\begin{smallmatrix} a & b_1 \\ c & d \end{smallmatrix}\right)$ and $\varepsilon = \det M$. Each assignment below that replaces a stored Euclidean quantity using a displayed linear relation is an in-place replacement of Lemma 5(3); that relation is the corresponding instance of (12). An empty block has $M = I$ and leaves the EEA data unchanged.

Figure 2 shows the early-to-late layout switch during a small-quotient block. Every coefficient of a replaced value is nonzero: $c > 0$, the remainders are positive, and $t, t' > 0$.

Lemma 8 (Storage and Uncomputation for Small Quotients). *For an interval input satisfying* (15) *and* (19)*, processing the small quotients fits the registers of* (7)*, uses $O(s_*(n))$ ancillas and $\widetilde{O}(n)$ Toffoli gates, and erases M and all temporary length and layout data before any large quotient is processed.*

Proof. Space. In the early layout the largest main pair created by the early branch of Algorithm 2 satisfies

$$rt' = cp + \varepsilon r' t \leq (c+1)p, \tag{24}$$

because $r't < Rt \leq p$; with (22) its combined length is $n + O(b)$. At a switch (Figure 2), $L \leq t < H_{\mathrm{sw}}$, so $U = BL < BH_{\mathrm{sw}}$ and $t' < (B+1)BH_{\mathrm{sw}}$: both t' and T' have at most $h + O(b)$ bits, and $R' < p/t' \leq 2^{n-h}$ fits the short register. Without a switch the short values stay below H_{sw}. In the late layout the remainders decrease, and $r't' < R't' \leq p$ and $R't' \leq p$ bound every main

Algorithm 2 Small-quotient phase of one coefficient interval.

Input: A state satisfying (15) and (19), stored in the early or late layout; interval $[L, U)$; layout data meta; sign η.

Output: The state after the maximal block of small quotients; M and all temporary length/layout registers zero.

Compute the transition matrix $M = \left(\begin{smallmatrix} a & b_1 \\ c & d \end{smallmatrix}\right)$ for the maximal block of small quotients; set $\varepsilon = \det M$ (Lemma 7).
if $M \neq I$ and the current layout is early $(r, T\,;\, t)$ **then**
    $T \leftarrow t'$ using $t' = at + cT$.
    $r \leftarrow r'$ using $t'r - \varepsilon t r' = cp$.
    $t \leftarrow T'$ using $\varepsilon t + cT' = dt'$. ▷ state $(r', t'\,;\, T')$
    if $t' < H_{\mathrm{sw}}$ **then**
        Exchange the main t' with the short T'. ▷ early layout $(r', T'\,;\, t')$
    else
        $T' \leftarrow R'$ using $r'T' + t'R' = p$.
        Exchange the main r' with the short R'. ▷ late layout $(R', t'\,;\, r')$
else if $M \neq I$ **then**
    $R \leftarrow r'$ using $r' = \varepsilon(ar - cR)$.
    $t \leftarrow t'$ using $\varepsilon r' t - r t' = -cp$.
    $r \leftarrow R'$ using $r - cR' = dr'$.
    Exchange the main r' with the short R'. ▷ late layout $(R', t'\,;\, r')$
Recover the pre-block lengths and layout from the output and M; update meta and clear the temporary controls.
$\eta \leftarrow \eta \oplus [\det M = -1]$.
Recover and erase M from the block output (Lemma 7).

pair of the late branch of Algorithm 2 and of the exchange. Each product bound holds with the old or the new target while the other subregister is unchanged, so it also bounds the largest target region. For example, $c < 2^{2b+1}$ gives

$$\mathrm{len}(r) + \mathrm{len}(t') \le n + 2b + 3, \qquad \mathrm{cap}_b(r) + \mathrm{cap}_b(t') < n + 4b + 3.$$

At a crossing from the early to the late layout, $t' < (B+1)BH_{\mathrm{sw}} < 2^{h+2b+1}$, so its rounded capacity is below $h + 3b + 1$; $T' < t'$ and $R' < 2^{n-h}$ fit as well. These bounds, together with the smaller bounds in the stored layouts, are covered by the $8b + 16$ slack in (7). Freed cells are zero.

Uncomputation. While the selected M is still retained, (21) recovers the old remainders and

$$t = \varepsilon(dt' - cT'), \qquad T = \varepsilon(-b_1 t' + aT') \tag{25}$$

the old coefficients. These expressions determine the old layout, the intermediate lengths, and the saved exchange-control bits, which are uncomputed while M is live; then η is toggled by $[\det M = -1]$ and M is erased by the oracle for Lemma 7(ii). An empty block uses $T' < L$ and the unchanged lengths.

Gates. Constantly many replacements, the two oracle calls from Lemma 7, and $\log^{O(1)} n$ controlled length and layout cases, each $\widetilde{O}(n)$ Toffoli gates in the same reusable workspace.

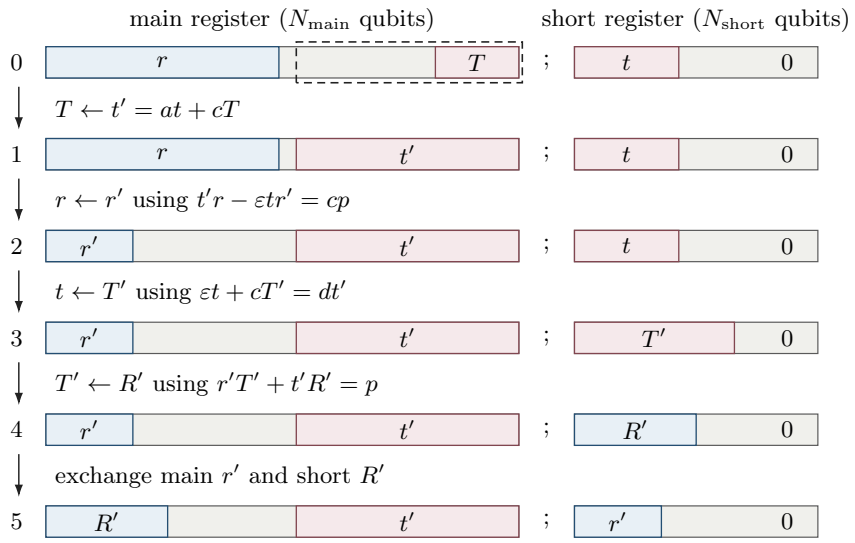

Fig. 2. A block of small quotients crossing $t < H_{\text{sw}} \leq t'$. Unchanged and exchanged values keep their widths; gray cells are zero. The dashed target region in the first row is the larger of its old and new lengths and is disjoint from r. Every row fits the two registers. M lives in separate $o(n)$ workspace until the cleanup of the length registers; the sign update and the erasure of M follow the last row.

4.4 Large Quotients

After the small quotients, compute the control bit

$$\chi = [r > 1]\,[t < U]\,[R \geq Br]. \tag{26}$$

For each shift

$$k = b(\lceil n/b \rceil - 1),\ b(\lceil n/b \rceil - 2),\ \ldots,\ 0, \tag{27}$$

execute one controlled digit update, including at inactive positions and leading zeros. On the active branch let (R_0, r, T_0, t) be the state before this division and $q = \lfloor R_0/r \rfloor$. If q_{proc} denotes the high part of q already processed, the current state is

$$A = R_0 - q_{\text{proc}} r, \qquad V = T_0 + q_{\text{proc}} t, \qquad At + rV = p. \tag{28}$$

The circuit represents q_{proc} through the current values A and V. The symbols A and V denote values in the existing layout: the layout chosen at entry is retained, so the early layout stores $(r, V\,;\, t)$ with A reconstructed from (28) and the late one stores $(A, t\,;\, r)$ with V reconstructed from (28), respectively. During digit processing, the invariant $At + rV = p$ is sufficient; the interval inequalities are restored when the division completes.

At position k, the next b-bit digit is

$$d_k = \left\lfloor \frac{A}{2^k r} \right\rfloor. \tag{29}$$

Since $q < 2^n$ and the unprocessed part of q is below 2^{k+b} at position k, $0 \le d_k < B$ throughout. The early layout adds $d_k 2^k t$ to the stored coefficient V; the late layout subtracts $d_k 2^k r$ from the stored remainder A. Algorithm 3 gives the complete reversible procedure, including cleanup.

Lemma 9 (Recovering a Quotient Digit). *After the update at shift k,*

$$d_k = \left\lfloor \frac{V_{\text{new}}}{2^k t} \right\rfloor \bmod B. \tag{30}$$

The digit and the temporary length and layout registers are uncomputed with $\widetilde{O}(n)$ Toffoli gates and $O(s_*(n))$ ancillas.

Proof. The processed part $q_{\text{proc}}^{\text{new}}$ is divisible by 2^k and has d_k as its b bits at position k, while $q_{\text{proc}}^{\text{old}}$ is divisible by 2^{k+b}. Since $0 \le T_0 < t$,

$$\left\lfloor \frac{T_0 + q_{\text{proc}}^{\text{new}} t}{2^k t} \right\rfloor = \frac{q_{\text{proc}}^{\text{new}}}{2^k},$$

and reducing modulo B isolates d_k; this includes $k = 0$ and $d_k = 0$. In the late layout, $V_{\text{new}} = (p - A_{\text{new}} t)/r$ is reconstructed from the stored values. Before erasing the digit, recover the old target as

$$V_{\text{old}} = V_{\text{new}} - d_k 2^k t, \qquad A_{\text{old}} = A_{\text{new}} + d_k 2^k r, \tag{31}$$

update meta, uncompute the temporary registers holding the old and maximum lengths, then erase d_k by (30). Every expression has constant depth and preserved inputs during its oracle call, so Corollary 1 applies.

Figure 3 shows one early digit update. During the digit updates the products stay bounded: for every processed part $q_{\text{proc}} \le q$, $A > 0$ and

$$rV \le r(T_0 + qt) < p, \qquad At \le p, \tag{32}$$

since $qr \le R_0$; the short value ($t < H_{\text{sw}}$ early, $r < 2^{n-h}$ late) is unchanged throughout the digit updates. Hence every intermediate layout fits (7).

After the last digit the successor state is

$$(R', r', T', t') = (r,\ R_0 - qr,\ t,\ T_0 + qt), \qquad R't' + r'T' = p.$$

The three restoration cases used in Algorithm 3 are forced by the layout at entry and whether t' crosses H_{sw}. In the early layout, $t'r + tr' = p$ replaces the main r by r'; if $t' \ge H_{\text{sw}}$, the additional identity $r't + t'R' = p$ moves the old $r = R'$ into the short register. In the late layout, $r't + rt' = p$ replaces the main t by t'. All coefficients in these replacements are nonzero. These completion steps also fit the registers of (7): in the early no-switch case, $t', T' < H_{\text{sw}}$ and both $R't'$ and $r't'$ are below p, bounding the old and new main target regions; at an early-to-late switch, $R't' < p$ bounds the enlarged main region and the temporary short

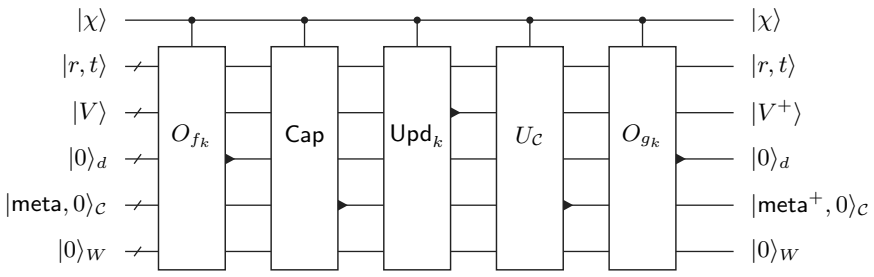

Fig. 3. One digit of a large quotient in the early layout. The blocks compute d_k, prepare the temporary length and layout data, update V, clear that data, and erase d_k using (30). All are controlled by χ and return the reusable workspace W to zero.

remainder satisfies $R' < p/t' \le 2^{n-h}$; in the late case, both remainders fit the short register and the relevant main products remain below p.

On the active branch, the final state determines its predecessor:

$$q = \lfloor t'/T' \rfloor, \qquad (R_0, r, T_0, t) = (qR' + r',\ R',\ t' - qT',\ T'). \tag{33}$$

These expressions recover the old layout and temporary lengths; they also recover the exchange controls, which are then cleared.

Lemma 10 (Recovering the Control Bit). *Assume the interval input satisfies* (15) *and* (19)*. On the final output of the interval,*

$$\chi = [L \le T' < U]\,[t' \ge BT']. \tag{34}$$

Proof. After a division with a large quotient, T' is its preceding $t \in [L, U)$ and $\lfloor t'/T' \rfloor = q \ge B$ by (23). If only a nonempty block of small quotients was processed, its last quotient satisfies $\lfloor t'/T' \rfloor < B$, that is $t' < (q_k + 1)T' \le BT'$. After an identity interval, $T' < L$. These cases are exhaustive; strict coefficient growth justifies both quotient recoveries.

Only the EEA data, the current layout, and η remain.

Example 1. For $p = 19$ and $x = 7$, we have $n = 5$, $b = 1$, $B = 2$, and $H_{\mathrm{sw}} = 8$, so a quotient is small if and only if it equals 1. Since $\mathrm{len}(R) \le n < \ell_{\mathrm{CF}} = 320$ throughout this example, matrix selection uses the materialized-input branch.

Initially $(R, r, T, t) = (19, 7, 0, 1)$. In the first interval $[1, 2)$ the small block is empty ($M = I$), and the next quotient is 2. After that large-quotient division the state is $(7, 5, 1, 2)$ at entry to $[2, 4)$.

Small block. The maximal small block consists of $q = 1$:

$$(R, r, T, t) = (7, 5, 1, 2) \longmapsto (5, 2, 2, 3), \qquad M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

Recovery from the output runs Euclid on $(t', T') = (3, 2)$: $3 = 1 \cdot 2 + 1$, and the smaller coefficient $1 < L = 2$ stops the recovery, giving back M.

Algorithm 3 Large-quotient phase of one coefficient interval.

Input: The state after the small-quotient block, the interval $[L, U)$, layout data meta, and sign η.

Output: The state after the optional large quotient; all digit, control, and temporary layout registers zero.

$\chi \leftarrow [r > 1][t < U][R \geq Br]$.
for $k = b(\lceil n/b \rceil - 1), \ldots, b, 0$ **do**
    $d_k \leftarrow \chi \lfloor A/(2^k r) \rfloor$ into a b-bit register.
    Compute the old, new, and maximum target lengths and the required layout controls.
    if the retained layout is early **then**
        $V \leftarrow V + d_k 2^k t$.
    else
        $A \leftarrow A - d_k 2^k r$.
    Recover the old target as $V_{\text{old}} = V_{\text{new}} - d_k 2^k t$ (early) or $A_{\text{old}} = A_{\text{new}} + d_k 2^k r$ (late); update meta and clear the temporary length and layout controls.
    Reconstruct V_{new} if necessary and erase $d_k \leftarrow d_k \oplus \big(\lfloor V_{\text{new}}/(2^k t) \rfloor \bmod B\big)$.
if $\chi = 1$ **then**
    if the retained layout is early and $t' < H_{\text{sw}}$ **then**
        $r \leftarrow r'$ using $t'r + tr' = p$; exchange the main t' with the short t. ▷ early layout $(r', T'\,;\, t')$, where $T' = t_{\text{old}}$
    else if the retained layout is early **then**
        $r \leftarrow r'$ using $t'r + tr' = p$.
        $t \leftarrow R'$ using $r't + t'R' = p$; exchange the main r' with the short R'. ▷ late layout $(R', t'\,;\, r')$
    else
        $t \leftarrow t'$ using $r't + rt' = p$; exchange the main r' with the short r. ▷ late layout $(R', t'\,;\, r')$, where $R' = r_{\text{old}}$
    $\eta \leftarrow \eta \oplus 1$.
    Recover the predecessor from $q = \lfloor t'/T' \rfloor$ and clear the completion controls and old-length data.
Erase $\chi \leftarrow \chi \oplus \big([L \leq T' < U][t' \geq BT']\big)$ from the interval output.

Large quotient. The next quotient is 2. Its nonzero bit is processed at $k = 1$:

$$(A, V) = (5, 2) \xrightarrow{d_1=1} (1, 8) \xrightarrow{d_0=0} (1, 8).$$

The updated state recovers both digits by Lemma 9:

$$\left\lfloor \frac{8}{2 \cdot 3} \right\rfloor \bmod 2 = 1, \qquad \left\lfloor \frac{8}{3} \right\rfloor \bmod 2 = 0.$$

Completion. The completed division has

$$(R', r', T', t') = (2, 1, 3, 8).$$

Since $t' = 8 = H_{\text{sw}}$, completion takes the early-to-late branch of Algorithm 3, giving the late layout $(2, 8\,;\, 1)$. The control bit is recovered from the interval

output as

$$[2 \le 3 < 4]\,[8 \ge 2 \cdot 3] = 1.$$

Output. Since $x = 7 < p/2$, we have $\eta_0 = 0$; after the three divisions with quotients $2, 1, 2$, $\eta = 1$. The inverter therefore returns $19 - 8 = 11$, and indeed $7 \cdot 11 \equiv 1 \pmod{19}$. Later intervals are identities because $r = 1$.

4.5 Output and Resource Bounds

At termination $r = 1$ and (15) gives

$$R = \lfloor p/t \rfloor, \qquad T = p \bmod t. \tag{35}$$

Preserve t and erase the other two stored subregisters by digitwise XOR of these expressions and of $r = 1$; their lengths are functions of t and are cleared too. Move t into the original target position by the controlled register permutations; vacated cells are zero, and t determines and clears the remaining length registers and layout bit. By Lemma 1, $1 = r \equiv (-1)^j tz$, so $z^{-1} \equiv (-1)^j t$; and $t < p/2$ since $R \ge 2$ and p is odd. Output t if $\eta = 0$ and $p - t$ otherwise: this is $x^{-1} \bmod p$ in both sign normalization cases. The output y recovers $\eta = [y > p/2]$, which clears the sign register.

The same construction can be controlled without changing the asymptotic cost. For an external control c, compute $a_c = c\,[1 \le x < p]$ and control every initialization, data, length, and layout update on a_c. On the inactive branch the evaluators use the constant tuple $(p, 1, 0, 1)$, which satisfies (15) and (19) for every $L \ge 1$; its windows are classically known constants and require no register. Branch-specific expressions, such as the division by T' in (33), use the default values of Section 3.1. Every evaluator preserves a_c and has the same measurement schedule on both branches. Finally erase a_c by $c\,[1 \le y < p]$, which agrees with its definition on both branches. This implements the controlled involution

$$J_p(x) = \begin{cases} x^{-1} \bmod p, & 1 \le x < p, \\ x, & x = 0 \text{ or } x \ge p. \end{cases} \tag{36}$$

Theorem 3 (Exact In-Place Modular Inversion). *The controlled permutation J_p has an exact implementation using*

$$\begin{aligned} &W_{\mathrm{inv}}(n) = \tfrac{3}{2}n + o(n) \text{ logical qubits, including the target,} \\ &G_{\mathrm{inv}}(n) = \widetilde{O}(n) \text{ Toffoli gates.} \end{aligned} \tag{37}$$

Proof. Lemma 6 gives termination with $r = 1$ after $\lceil n/b \rceil + 1$ intervals, and Section 4.5 gives the output. Each interval uses constantly many replacements and the two oracle calls of Lemma 7 for the small quotients, plus $\lceil n/b \rceil$ digit positions and constantly many oracle calls that erase the temporary data of a possible large quotient. With the scans over possible lengths, constants C_{inv} and D_{inv} give

$$\begin{aligned} G_{\mathrm{inv}}(n) &\le C_{\mathrm{inv}}\, n \log^{D_{\mathrm{inv}}}(n+2)\,(n/b+1)^2 \\ &= O\big(n \log^{2K+D_{\mathrm{inv}}}(n+2)\big) = \widetilde{O}(n), \end{aligned} \tag{38}$$

where D_{inv} absorbs the constant exponents in the oracle, continued-fraction, and length-scan bounds. The width is

$$N_{\mathrm{main}} + N_{\mathrm{short}} + O(s_*(n)) = n + h + O(s_*(n)) = \frac{3}{2}n + o(n) : \tag{39}$$

the target is part of the main register. Beyond the main and short registers, a reusable $O(s_*(n))$-qubit workspace holds the evaluator and simulation state, retained continued-fraction wires, temporary buffers, length and layout metadata, and at most one $O(b)$-bit transition matrix or quotient digit. The matrix M is erased before the first digit is created, so these two $O(b)$-bit objects are never live simultaneously. The controls η, χ, and a_c use $O(1)$ additional qubits. Since $b, \log n = O(s_*(n))$, a constant multiple of $s_*(n)$ suffices throughout; all measurement records are classical.

For exactness, every temporary value can be recomputed from the updated state: M by Lemma 7, quotient digits by (30), the temporary length and layout data by (25), (31), and (33), and χ by (34). Each erasure uses an oracle whose inputs are preserved for that call. Definition 1 then composes the resulting exact implementations to satisfy (9), with all ancillas returned to zero.

5 Affine Point Addition

Point addition combines modular inversion, constant multiply-adds, and multiplication by a variable square. The variable-square map is expressed through inversions of Theorem 3; each inversion preserves the other coordinate and reuses the same $h + O(s_*(n))$-qubit workspace. A final permutation on a constant number of encodings supplies the exceptional cases of the group law.

5.1 Field Operations

The coordinate maps below are over $\mathbb{F}_p$; the circuits use canonical representatives. For a classical constant $c_{\mathrm{ma}} \in \mathbb{F}_p$, the modular multiply-add

$$(X, Y) \longmapsto (X,\ Y + c_{\mathrm{ma}} X \bmod p)$$

uses $\widetilde{O}(n)$ Toffoli gates and $o(n)$ ancillas: generate the digits of the expression $f(X) = c_{\mathrm{ma}} X \bmod p$ from the preserved X and add them to an $(n+1)$-bit extension of Y by Lemma 5(1); the sum is below $2p$; compare it with p and conditionally subtract p. The reduction bit is recovered as

$$e_{\mathrm{red}} = [Y_{\mathrm{out}} < f(X)], \tag{40}$$

since without reduction the output is at least $f(X)$ and with reduction it is smaller; the extension bit is then zero. Guard the operation by $[X < p][Y < p]$ and use the identity outside the pairs of canonical field elements. The guard is recovered from the output because the canonical set and its complement are invariant. Constant additions, negation, and swaps obey the same bounds.

For a classical $\delta \neq 0$, define the scaled swap S_δ by

$$S_\delta(X, Y) = (\delta Y,\ \delta^{-1} X). \tag{41}$$

It is implemented by three multiply-adds followed by a negation: $X \leftarrow X + \delta Y$, $Y \leftarrow Y - \delta^{-1} X$, $X \leftarrow X + \delta Y$, $Y \leftarrow -Y$. If $\mathsf{Swap}(X, Y) = (Y, X)$ denotes the ordinary swap, then

$$S_\delta \circ \mathsf{Swap} : (X, Y) \longmapsto (\delta X,\ \delta^{-1} Y).$$

Thus diagonal scaling is obtained by applying the ordinary swap first and S_δ second.

The remaining primitive is multiplication by a variable square. Let $J(0) = 0$ and $J(x) = x^{-1}$ for $x \neq 0$ in $\mathbb{F}_p$; this is J_p on canonical values. Hua's identity [12],

$$a - \left(a^{-1} + (b^{-1} - a)^{-1}\right)^{-1} = aba \qquad \text{whenever all inverses exist,}$$

expresses a product through inversions and subtractions. Reading it with $a = X$ and $b = Y$ gives the following.

Lemma 11 (Multiplication by a Square). *There is a permutation $\mathcal{D}$ of $\mathbb{F}_p^2$ with*

$$\mathcal{D}(X, Y) = (X,\ X^2 Y) \qquad (X \neq 0). \tag{42}$$

Both $\mathcal{D}$ and $\mathcal{D}^{-1}$ use five sequential calls to J, constantly many additions and subtractions, and one correction on two values of Y.

Proof. Apply, in order,

$$\begin{aligned} &Y \leftarrow J(Y), \quad Y \leftarrow Y - X, \quad Y \leftarrow J(Y), \quad X \leftarrow J(X), \\ &Y \leftarrow Y + X, \quad Y \leftarrow J(Y), \quad X \leftarrow J(X), \quad Y \leftarrow X - Y. \end{aligned} \tag{43}$$

For $X \neq 0$ and $Y \notin \{0, X^{-1}\}$ the successive second coordinates are

$$\frac{1}{Y}, \quad \frac{1 - XY}{Y}, \quad \frac{Y}{1 - XY}, \quad \frac{1}{X(1 - XY)}, \quad X(1 - XY), \quad X^2 Y,$$

and the first coordinate returns to X. The inputs $Y = 0$ and $Y = X^{-1}$ produce second-coordinate outputs X and 0, respectively, so exchanging these two outputs gives (42) on all of Y for $X \neq 0$: compute $[Y \in \{0, X\}]$, conditionally XOR the encoding of X into Y, and uncompute the predicate, whose set is invariant. For $X = 0$ the correction is the identity and the sequence gives $Y \mapsto -J(Y)$, also a permutation, so $\mathcal{D}$ is total. Reversing the logical sequence with fresh oracle calls gives $\mathcal{D}^{-1}$.

Figure 4 implements (43): J_X, J_Y invert the indicated coordinate, $\mathsf{Sh}_\pm$ maps $Y \leftarrow Y \pm X$, N_Y maps $Y \leftarrow X - Y$, and $\mathsf{Exch}_{0,X}$ performs the exceptional exchange. Each inversion reuses the same $h + O(s_*(n))$-qubit workspace W and returns it to zero; reversing the logical sequence with fresh oracle calls implements $\mathcal{D}^{-1}$.

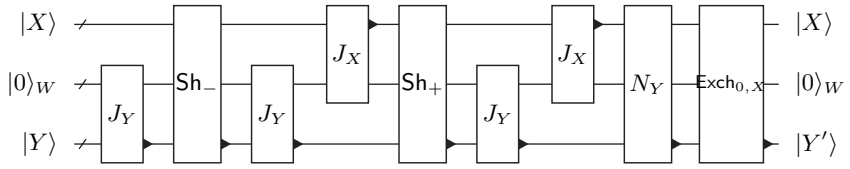

Fig. 4. Multiplication by a variable square using five inversions. Each inversion acts on one coordinate and the shared workspace W, returning W to zero. The final exchange corrects the exceptional values, giving $Y' = X^2Y$ for $X \neq 0$ and $Y' = -J(Y)$ for $X = 0$.

5.2 Addition by a Constant Point

If $Q_0 = \mathcal{O}$, controlled addition is the identity. Henceforth assume $Q_0 = (u, v)$ is affine. First suppose $v \neq 0$; the two-torsion case $v = 0$ is treated below. Put

$$\delta = 2v, \qquad \gamma = (3u^2 + a_E)/\delta, \tag{44}$$

and write $\Delta = x - u$, $\Delta_+ = x_+ - u$ for the input and output coordinate differences. The chord formulas imply

$$\frac{\Delta_+}{\delta} = \frac{v - y + \gamma\Delta}{\Delta^2},$$

so $\mathcal{D}^{-1}$ turns the prepared second coordinate $v - y + \gamma\Delta$ into Δ_+/δ; a scaled swap then gives $(\Delta_+, \Delta/\delta)$, and $\mathcal{D}$ supplies the square factor needed for the output y-coordinate. This motivates the permutation

$$\begin{aligned}
&X \leftarrow X - u, \quad Y \leftarrow v - Y, \quad Y \leftarrow Y + \gamma X,\\
&(X, Y) \leftarrow \mathcal{D}^{-1}(X, Y),\\
&(X, Y) \leftarrow (\delta Y,\ \delta^{-1} X),\\
&(X, Y) \leftarrow \mathcal{D}(X, Y),\\
&Y \leftarrow Y - v - \gamma X, \quad X \leftarrow X + u.
\end{aligned} \tag{45}$$

The derivation applies to affine points with $\Delta, \Delta_+ \neq 0$. The input difference vanishes at $\pm Q_0$; an output with $x_+ = u$ arises from $\mathcal{O}$ or $-2Q_0$; and $-Q_0$ maps to $\mathcal{O}$. These form the exceptional set below. The coordinate permutation itself acts for either value of the infinity flag and leaves the flag unchanged; the final correction handles its action on $\operatorname{enc}\mathcal{O}$.

Lemma 12 (Nonexceptional Points). *For $P \notin \{\mathcal{O}, Q_0, -Q_0, -2Q_0\}$, the permutation* (45) *maps* $\operatorname{enc} P$ *to* $\operatorname{enc}(P + Q_0)$.

Proof. Both differences are nonzero. Subtracting the curve equations at x and u gives $y^2 - v^2 = \Delta(3u^2 + a_E) + 3u\Delta^2 + \Delta^3$, and (2) gives $\Delta_+ = (y - v)^2/\Delta^2 - \Delta - 3u$; together,

$$\frac{\Delta_+}{\delta} = \frac{\gamma}{\Delta} + \frac{v - y}{\Delta^2}, \qquad \frac{\Delta\Delta_+}{\delta} = \gamma - \frac{y - v}{\Delta}. \tag{46}$$

The coordinate pairs after preparation, $\mathcal{D}^{-1}$, the scaled swap, and $\mathcal{D}$ are

$$(\Delta,\ v - y + \gamma\Delta), \quad (\Delta,\ \Delta_+/\delta), \quad (\Delta_+,\ \Delta/\delta), \quad (\Delta_+,\ \Delta\Delta_+^2/\delta),$$

and the final updates give $X = x_+$ and $Y = \Delta\Delta_+^2/\delta - v - \gamma\Delta_+ = -v - \frac{y-v}{\Delta}\Delta_+ = y_+$ by the chord formula.

It remains to handle a two-torsion addend. For $Q_0 = (u, 0)$ put $\beta_0 = 3u^2 + a_E$, which is nonzero because the curve is nonsingular at a point with $y = 0$. Consider the sequence

$$\begin{aligned} &X \leftarrow X - u, \quad X \leftarrow J(X), \\ &(X, Y) \leftarrow (\beta_0 X,\ \beta_0^{-1} Y), \quad (X, Y) \leftarrow \mathcal{D}(X, Y), \\ &Y \leftarrow -Y, \quad X \leftarrow X + u \end{aligned} \tag{47}$$

where the diagonal scaling in the second line is implemented as $S_{\beta_0} \circ \mathsf{Swap}$, namely by the ordinary swap followed by the scaled swap of (41). This sequence agrees with addition of Q_0 outside $\{\mathcal{O}, Q_0\}$: for $\Delta \neq 0$, the chord formula and $u^3 + a_E u + b_E = 0$ give $\Delta_+ = \beta_0/\Delta$ and $y_+ = -\beta_0 y/\Delta^2$. The remaining inputs $\mathcal{O}$ and Q_0 are handled by the exceptional correction below.

5.3 Exceptional Cases

Let Φ_0 be (45) or (47), extended to preserve the infinity flag, and let $\mathcal{E}$ be its exceptional set with duplicates removed. Classically compute

$$g_s = \Phi_0(\mathrm{enc}\, s), \qquad t_s = \mathrm{enc}(s + Q_0), \qquad s \in \mathcal{E}. \tag{48}$$

Lemma 13 (Correction for Exceptional Points). *A permutation* Corr *supported on at most eight strings makes* $\mathsf{Corr} \circ \Phi_0$ *agree with addition of* Q_0 *on all canonical point encodings. The controlled correction has an exact implementation using* $\widetilde{O}(n)$ *Toffoli gates and* $o(n)$ *ancillas.*

Proof. The two lists are injective and disjoint from the common image of the nonexceptional points. Their edges $g_s \mapsto t_s$ form disjoint paths and cycles. Close each path from its last vertex to its first, giving a permutation of their union, which has size at most $2|\mathcal{E}| \leq 8$, and use the identity elsewhere. Compute the input's support index, apply the XOR difference for its edge, and erase the index using the inverse support table on the output. Constantly many equality tests and controlled XORs satisfy the bounds; include the external control in the support test.

Figure 5 shows the full controlled circuit. The first and last blocks are the affine maps $A_{\mathrm{in}}(X, Y) = (X - u,\ v - Y + \gamma(X - u))$ and $A_{\mathrm{out}}(X, Y) = (X + u,\ Y - v - \gamma X)$, S_δ is (41), and Corr the exceptional correction. During each inversion the coordinate being inverted is extended by $8b + 16$ workspace cells

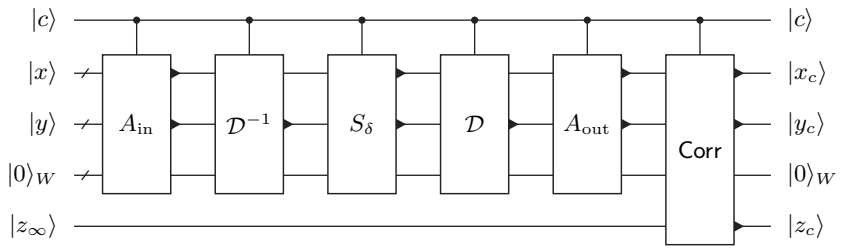

Fig. 5. Controlled addition of $Q_0 = (u, v)$, $v \neq 0$. Only Corr changes the infinity flag. Each of $\mathcal{D}^{-1}$ and $\mathcal{D}$ uses five inversions, and every block returns the shared workspace W to zero.

to form the main register of Section 4; the shared workspace W of $h + O(s_*(n))$ qubits contains these cells, the short register, and the $O(s_*(n))$ ancillas used by the window and predicate oracles, and excludes both coordinates, the control, and the flag.

Proof (Theorem 2). The case $Q_0 = \mathcal{O}$ is the identity. Otherwise, the generic sequence uses ten inversions, five in each of $\mathcal{D}^{-1}$ and $\mathcal{D}$; the two-torsion sequence uses six. Every other block costs $\widetilde{O}(n)$ Toffoli gates and $o(n)$ ancillas. Since each inversion preserves the other n-qubit coordinate,

$$\begin{aligned} W_{\text{add}}(n) &= n + W_{\text{inv}}(n) + O(1) = \tfrac{5}{2}n + o(n), \\ G_{\text{add}}(n) &\leq 10\, G_{\text{inv}}(n) + \widetilde{O}(n) = \widetilde{O}(n). \end{aligned} \tag{49}$$

The flag and the control use $O(1)$ qubits, and each field-operation block returns canonical coordinate residues and zero workspace before the next block, so Definition 1 composes, including with a reference system.

6 Discrete Logarithm Algorithm

We use the eigenvalue-estimation form of Shor's algorithm [16,22] with one point register and one reusable control qubit, and recover the subgroup order in classical postprocessing. Ekerå [6] studies related algorithms and trade-offs without prior knowledge of the group order.

6.1 Semiclassical Phase Estimation

The cases $G = \mathcal{O}$ or $Q = \mathcal{O}$ return $d = 0$; the promise forces $Q = \mathcal{O}$ when $G = \mathcal{O}$. Otherwise let $r_G = \operatorname{ord}(G)$ and $Q = dG$. Since each $x \in \mathbb{F}_p$ has at most two corresponding y-coordinates,

$$r_G \leq \#E(\mathbb{F}_p) \leq 2p + 1 < 2^{n+2}, \tag{50}$$

which suffices for rational recovery. Addition of G and of Q have the common eigenvectors

$$|\psi_k\rangle = \frac{1}{\sqrt{r_G}} \sum_{j=0}^{r_G-1} e^{-2\pi i k j/r_G} |\mathrm{enc}(jG)\rangle, \qquad 0 \le k < r_G, \tag{51}$$

with eigenphases k/r_G and $(kd \bmod r_G)/r_G$. Initially $|\mathrm{enc}\,\mathcal{O}\rangle = r_G^{-1/2} \sum_k |\psi_k\rangle$. Each phase-estimation procedure is diagonal in this common eigenbasis and preserves every component $|\psi_k\rangle$. Orthogonality therefore gives the joint distribution obtained by first choosing a uniform k and then sampling both phase-estimation outcomes for that k. We call the pair a *joint sample*.

Lemma 14 (Joint Samples). *For any constant $\xi > 0$, one joint sample recovers $\theta = k/r_G$ and $\phi = (kd \bmod r_G)/r_G$ exactly with probability at least $1 - 2\xi$, uniformly in k, using $O(n)$ controlled point additions, one reusable control qubit, and polynomial classical time.*

Proof. Use $m = 2(n+2) + c_\xi$ phase bits. For a sufficiently large constant c_ξ, the tail bound of phase estimation gives circular error below $1/(2 \cdot 2^{2(n+2)})$ with probability at least $1 - \xi$, and continued fractions then recover the unique phase with denominator below 2^{n+2}, treating the endpoints modulo one; a union bound covers both estimates. The $2m$ controlled powers add the precomputed points $2^j G$ and $2^j Q$; the semiclassical inverse QFT [8] reuses the measured control with rotations set by the earlier outcomes, as specified in Appendix E.1. Classical doubling and rational reconstruction take polynomial time.

6.2 Classical Postprocessing

Take two independent joint samples (θ_i, ϕ_i), resetting the point register between them, and let $\widehat{r}$ be the lcm of the denominators of θ_1, θ_2. For correct samples, $\widehat{r} = r_G$ exactly when $\gcd(k_1, k_2, r_G) = 1$, an event of probability at least $\zeta(2)^{-1} > 0.6$. Then the extended Euclidean algorithm on $k_1, k_2, \widehat{r}$ recovers d from $\widehat{r}\phi_1, \widehat{r}\phi_2$. We return a candidate only after checking $dG = Q$; Appendix E.2 gives the formulas.

Proof (Theorem 1). Take $\xi = 1/100$. A union bound gives success probability at least $\zeta(2)^{-1} - 4/100 > 1/2$ per attempt, so two attempts succeed with probability above $3/4$. By Theorem 2, the $O(n)$ controlled additions use $\frac{5}{2}n + o(n)$ qubits and $\widetilde{O}(n^2)$ Toffoli gates, with workspace reused. All classical processing is polynomial time. The inverse QFT uses $O(n)$ one-qubit rotations; Clifford+T synthesis to inverse-polynomial accuracy [5] changes the output distribution by less than $1/24$ and adds $\widetilde{O}(n)$ Clifford+T gates, no Toffoli gates, and no ancillas. The success probability remains above $2/3$.

7 Conclusion

We obtain exact in-place prime-field inversion with $3n/2 + o(n)$ logical qubits and $\widetilde{O}(n)$ Toffoli gates, leading to ECDLP with $5n/2 + o(n)$ qubits and $\widetilde{O}(n^2)$ Toffoli gates.

The polylogarithmic factors and lower-order space terms are not optimized. It remains open whether exact in-place inversion can achieve $\widetilde{O}(n)$ Toffoli gates with $n + o(n)$ qubits, and whether the same storage strategy extends to other Euclidean domains.

AI Disclosure. OpenAI GPT-5.6 Sol (Pro) was used in the discovery and development of the modular-inversion and point-addition constructions presented in this paper. The authors posed the problem, directed the exploration, and selected among candidate approaches. OpenAI Codex was used for the LaTeX source and for numerical checks of the worked example and the arithmetic identities. Claude Fable 5.1 and GPT-6 Astra were used in later revisions for literature review, proof auditing, exposition, and consistency checks; references located by these tools were checked by the authors against the cited sources, and errors identified in the audits were corrected. The authors read and verified every definition, lemma, and proof in the final version and take full responsibility for the paper's contents and correctness.

References

1. Babbush, R., Zalcman, A., Gidney, C., Broughton, M., Khattar, T., Neven, H., Bergamaschi, T., Drake, J., Boneh, D.: Securing elliptic curve cryptocurrencies against quantum vulnerabilities: Resource estimates and mitigations. PRX Quantum **7**, 031001 (2026). https://doi.org/10.1103/j3xf-bw18
2. Bennett, C.H.: Logical reversibility of computation. IBM Journal of Research and Development **17**(6), 525–532 (1973). https://doi.org/10.1147/rd.176.0525
3. Brent, R.P., Zimmermann, P.: Modern Computer Arithmetic. Cambridge University Press (2010). https://doi.org/10.1017/CBO9780511921698
4. Chevignard, C., Fouque, P.-A., Schrottenloher, A.: Reducing the Number of Qubits in Quantum Discrete Logarithms on Elliptic Curves. In: EUROCRYPT 2026. pp. 371–401. Springer (2026). https://doi.org/10.1007/978-3-032-25291-3_13. Full version, 27 May 2026: https://inria.hal.science/hal-05634621v1
5. Dawson, C.M., Nielsen, M.A.: The Solovay–Kitaev algorithm. Quantum Information & Computation **6**(1), 81–95 (2006). https://arxiv.org/abs/quant-ph/0505030
6. Ekerå, M.: Quantum algorithms for computing general discrete logarithms and orders with tradeoffs. Journal of Mathematical Cryptology **15**(1), 359–407 (2021). https://doi.org/10.1515/jmc-2020-0006
7. Gidney, C.: Spooky pebble games and irreversible uncomputation. Blog post (2019). https://algassert.com/post/1905
8. Griffiths, R.B., Niu, C.-S.: Semiclassical Fourier transform for quantum computation. Physical Review Letters **76**(17), 3228–3231 (1996). https://doi.org/10.1103/PhysRevLett.76.3228
9. Häner, T., Jaques, S., Naehrig, M., Roetteler, M., Soeken, M.: Improved quantum circuits for elliptic curve discrete logarithms. In: PQCrypto 2020, LNCS vol. 12100, pp. 425–444. Springer (2020). https://doi.org/10.1007/978-3-030-44223-1_23
10. Häner, T., Tripier, F., Young, J., Naehrig, M., Maksymov, A., Alam, S., Maslov, D., Parrott, M., de Sereville, Y., Sullivan, J., Webster, M., Delfosse, N., Gamble, J., Roetteler, M.: Computing 256-bit elliptic curve discrete logarithms in 26 days on a fault-tolerant trapped-ion quantum computer with 20,000 qubits.

arXiv:2609.05625 (2026); Cryptology ePrint Archive, Paper 2026/1916. https://doi.org/10.48550/arXiv.2609.05625
11. Harvey, D., van der Hoeven, J.: Integer multiplication in time $O(n \log n)$. Annals of Mathematics **193**(2), 563–617 (2021). https://doi.org/10.4007/annals.2021.193.2.4
12. Hua, L.-K.: On the automorphisms of a sfield. Proceedings of the National Academy of Sciences of the United States of America **35**(7), 386–389 (1949). https://doi.org/10.1073/pnas.35.7.386
13. Kaye, P., Zalka, C.: Optimized quantum implementation of elliptic curve arithmetic over binary fields. Quantum Information & Computation **5**(6), 474–491 (2005). https://arxiv.org/abs/quant-ph/0407095
14. Khattar, T., Shutty, N., Gidney, C., Zalcman, A., Yosri, N., Maslov, D., Babbush, R., Jordan, S.P.: Verifiable quantum advantage via optimized DQI circuits. arXiv:2510.10967 (2025). https://arxiv.org/abs/2510.10967
15. Kim, H., Jang, K., Wang, S., Srivastava, V., Baksi, A., Song, G., Seo, H., Chattopadhyay, A.: New Quantum Circuits for ECDLP: Breaking Prime Elliptic Curve Cryptography. Cryptology ePrint Archive, Paper 2026/106 (2026). Version dated 26 May 2026. https://eprint.iacr.org/2026/106
16. Kitaev, A.Y.: Quantum measurements and the Abelian stabilizer problem. arXiv:quant-ph/9511026 (1995). https://arxiv.org/abs/quant-ph/9511026
17. Kornerup, N., Sadun, J., Soloveichik, D.: Tight Bounds on the Spooky Pebble Game: Recycling Qubits with Measurements. Quantum **9**, 1636 (2025). https://doi.org/10.22331/q-2025-02-18-1636
18. Litinski, D.: How to compute a 256-bit elliptic curve private key with only 50 million Toffoli gates. arXiv:2306.08585 (2023). https://arxiv.org/abs/2306.08585
19. Long, J., Pender, T., Huang, Z., Santos, M.B., Singh, S.K., Naskręcki, B., et al.: ECDSA.Fail: Open autoresearch for optimizing elliptic-curve point addition in Shor's algorithm. arXiv:2609.09582v1 (2026), 9 September 2026. https://doi.org/10.48550/arXiv.2609.09582
20. Luo, H., Yang, Z., Luo, J., Wang, Z., Su, Y., Sun, X., Li, L., Li, T.: Quantum algorithm for elliptic curve discrete logarithms with space-efficient point addition. arXiv:2607.13816v3 (2026), 5 September 2026. https://doi.org/10.48550/arXiv.2607.13816
21. Möller, N.: On Schönhage's algorithm and subquadratic integer GCD computation. Mathematics of Computation **77**(261), 589–607 (2008). https://doi.org/10.1090/S0025-5718-07-02017-0
22. Mosca, M., Ekert, A.: The hidden subgroup problem and eigenvalue estimation on a quantum computer. In: QCQC 1998, LNCS vol. 1509, pp. 174–188. Springer (1999). https://doi.org/10.1007/3-540-49208-9_15
23. Pippenger, N., Fischer, M.J.: Relations among complexity measures. Journal of the ACM **26**(2), 361–381 (1979). https://doi.org/10.1145/322123.322138
24. Proos, J., Zalka, C.: Shor's discrete logarithm quantum algorithm for elliptic curves. Quantum Information & Computation **3**(4), 317–344 (2003). https://doi.org/10.26421/QIC3.4-3
25. Roetteler, M., Naehrig, M., Svore, K.M., Lauter, K.: Quantum resource estimates for computing elliptic curve discrete logarithms. In: ASIACRYPT 2017, LNCS vol. 10625, pp. 241–270. Springer (2017). https://doi.org/10.1007/978-3-319-70697-9_9
26. Schönhage, A.: Schnelle Berechnung von Kettenbruchentwicklungen. Acta Informatica **1**, 139–144 (1971). https://doi.org/10.1007/BF00289520

27. Schrottenloher, A.: Optimized point addition circuits for elliptic curve discrete logarithms. arXiv:2606.02235 (2026). https://arxiv.org/abs/2606.02235
28. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172

Supplementary Material

The following appendices give supporting proofs, implementation details, and additional diagrams. The main construction and resource arguments are in Sections 3–6.

A Arithmetic on Materialized Operands

We prove Lemma 2. The multiplication algorithm is an ordinary Boolean computation using fixed-point Fourier transforms. Its precision is chosen so that the final integer coefficients are recovered exactly. The recurrences for division and inversion modulo a power of two then provide exact corrections at every precision.

For the division and odd-inversion resource bounds, evaluate the recurrences below down to constant precision, completing each lower-precision call before allocating the temporary storage of its caller. Keeping one copy of the divisor and reusing the current and output buffers leaves $O(m)$ bits live. There are $O(\log(m+2))$ precision levels, with a constant number of multiplications and comparisons at each; sign separation handles signed arithmetic, including the floor correction for division, and variable shifts use an oblivious barrel-shift network. Thus, once the exact corrections below are established, these operations have $m \log^{O(1)}(m+2)$ work and $O(m)$ working bits.

A.1 Product Digits

Algorithm 4 is the column scan used in Section 3.1. Its operands are represented implicitly; only the two current w-bit digits are stored.

Algorithm 4 One digit of a product of implicitly represented integers.

Input: Nonnegative operands represented by expressions U, V; digit index j.
Output: $\lfloor UV/2^{jw} \rfloor \bmod 2^w$ in a w-bit output buffer.

$C \leftarrow 0$ ▷ carry from the preceding column
for $k = 0, \ldots, j$ **do**
  $S \leftarrow C$
  for $i = 0, \ldots, k$ **do**
    $u \leftarrow \mathsf{Query}(U, iw, w)$; $v \leftarrow \mathsf{Query}(V, (k-i)w, w)$
    $S \leftarrow S + uv$ ▷ materialized digits; $O(w)$ working bits
    reuse the buffers of u, v
  $C \leftarrow \lfloor S/2^w \rfloor$
return $S \bmod 2^w$

The accumulator has $2w + O(\log g)$ bits. Queries beyond an operand's length return zero. Both loops run to the input-independent bound $O(g)$; the index j enters only as a control.

A.2 Multiplication

Proof (Multiplication part of Lemma 2). For multiplication, use chunks of $d = \lceil \log_2(m+2) \rceil$ bits. Choose a power-of-two Fourier length $N_{\mathrm{FFT}} = O(m/\log(m+2))$ sufficient for their linear convolution. Chunk magnitudes are $O(m)$; forward transform values, pointwise products, and unnormalized inverse-transform intermediates are all bounded by $(m+2)^{O(1)}$.

Use $C_1 \lceil \log_2(m+2) \rceil + C_2$ fractional bits and $C_3 \lceil \log_2(m+2) \rceil + C_4$ integer bits per real component, for sufficiently large absolute constants. These precisions can be chosen uniformly: each butterfly has bounded norm, there are $O(\log m)$ stages, and the magnitudes of both exact and rounded intermediates are polynomially bounded. More explicitly, if E_j is the maximum component error at a transform stage and P_{frac} is the fractional precision, its recurrence has the form

$$E_{j+1} \le (2 + O(2^{-P_{\mathrm{frac}}}))E_j + (m+2)^{C_5} 2^{-P_{\mathrm{frac}}},$$

with an absolute C_5. The same bound applies to the inverse transform, while one pointwise multiplication amplifies input error by only another polynomial factor. Increasing C_1 makes the final coefficient error less than $1/4$ for every input. Rounding recovers the exact integer convolution; carry normalization completes the product. Twiddle approximations are input-independent constants computed to this precision. This is the standard reduction of integer multiplication to fixed-point convolution [3].

There are $O(m/\log(m+2))$ fixed-point words of $O(\log(m+2))$ bits, including the transform arrays and other working memory. Their total size is $O(m)$. Schoolbook operations on these small words give $m \log^{O(1)}(m+2)$ Boolean operations, and all transform addresses are input-independent.

A.3 Reciprocals and Division

For $D > 0$, put $\ell_D = \lfloor \log_2 D \rfloor$ and define

$$R_m(D) = \left\lfloor \frac{2^{\ell_D+m}}{D} \right\rfloor, \qquad m \geq 1. \tag{52}$$

Above the base precision, let $m' = \lceil m/2 \rceil$ and set

$$y = 2^{m-m'} R_{m'}(D), \qquad \tau = 2y - \left\lceil \frac{Dy^2}{2^{\ell_D+m}} \right\rceil. \tag{53}$$

Then the exact correction is

$$R_m(D) = \tau + [D(\tau+1) \leq 2^{\ell_D+m}] + [D(\tau+2) \leq 2^{\ell_D+m}]. \tag{54}$$

Indeed, write $a = D/2^{\ell_D} \in [1,2)$, $\mathcal{R} = 2^m/a$, and $e = \mathcal{R} - y$. The induction hypothesis gives $0 \leq e < 2^{m-m'}$, so

$$2y - \frac{ay^2}{2^m} = \mathcal{R} - \frac{ae^2}{2^m}, \qquad 0 \leq \frac{ae^2}{2^m} < 2^{m-2m'+1} \leq 2.$$

Thus $\lfloor \mathcal{R} \rfloor - \tau \in \{0,1,2\}$, and the two tests in (54) select the correct integer.

Base precision. When $m \leq w$, define

$$e_D = \max\{0, \ell_D + 1 - (m+2)\}, \quad D_0 = \lfloor D/2^{e_D} \rfloor, \quad q_0 = \left\lfloor \frac{2^{\ell_D+m-e_D}}{D_0} \right\rfloor. \tag{55}$$

The base-case numerator and denominator have $O(w)$ bits, and

$$R_m(D) = q_0 - [Dq_0 > 2^{\ell_D+m}]. \tag{56}$$

If $e_D = 0$, the computation is exact already. Otherwise $D_0 \geq 2^{m+1}$, and replacing D by $D_0 2^{e_D}$ increases the real reciprocal by less than $2^m/D_0 \leq 1/2$. The integer overestimate is therefore at most one. The correcting comparison uses a digit scan over D.

Division. For $0 \leq A < 2^{N_{\text{num}}}$, take $m = N_{\text{num}} + 2$ and evaluate the required bits of

$$q_0 = \left\lfloor \frac{AR_m(D)}{2^{\ell_D+m}} \right\rfloor.$$

Its underlying real value underestimates A/D by less than $1/4$. Consequently

$$\lfloor A/D \rfloor = q_0 + [D(q_0+1) \leq A]. \tag{57}$$

The remainder is $A - D\lfloor A/D \rfloor$. All division corrections in Lemma 3 are therefore comparisons implemented by digit scans.

A.4 Inversion Modulo a Power of Two

For odd D, let $V_{m'}D \equiv 1 \pmod{2^{m'}}$ with $m' = \lceil m/2 \rceil$. Then

$$V_m = V_{m'}(2 - DV_{m'}) \bmod 2^m \tag{58}$$

is the exact inverse modulo 2^m, because

$$1 - DV_m \equiv (1 - DV_{m'})^2 \equiv 0 \pmod{2^m}.$$

At precision $m \leq w$, only the low $O(w)$ bits of the divisor are needed. For a data-dependent modulus 2^ℓ below a known bound, compute the required low bits at the common precision and mask them at ℓ. The precision schedule is input-independent.

Together with the multiplication argument and the resource analysis above, the exact reciprocal, division, and odd-inverse identities complete the proof of Lemma 2.

B In-Place Integer Updates

We prove Lemma 5. All generated digits in this appendix are supplied and uncomputed by the oracles of Corollary 1. Each oracle call includes all phase corrections.

B.1 Addition

Split the target into chunks of at most w bits. For a chunk of actual width w_j, generate the addend digit a_j from input registers that remain unchanged during the call and compute

$$c_{j+1} = [y_j + a_j + c_j \geq 2^{w_j}], \qquad y'_j = (y_j + a_j + c_j) \bmod 2^{w_j}, \tag{59}$$

starting with $c_0 = 0$. The chunk permutation uses an $O(w)$-bit temporary buffer: compute its output, swap it with the target, and uncompute the old value using the inverse chunk map. Uncompute a_j from the unchanged inputs.

After all chunks, uncompute the carries in descending order using

$$c_{j+1} = [y'_j < a_j + c_j]. \tag{60}$$

At this point c_j is still retained. The comparison has $w_j + 1$ bits, so it covers the case $a_j + c_j = 2^{w_j}$. The final overflow bit is uncomputed in the same way, giving an in-place addition modulo the target's full power of two. There are $O(n/w) = O(g)$ digits and carries. The temporary registers use $O(w + g)$ ancilla qubits in addition to the reusable $O(s_*(n))$ ancillas of Corollary 1, and the Toffoli count is $\widetilde{O}(n)$. For subtraction, use the digits of the negated operand reduced modulo 2^m. An input may occupy another part of the same register, provided that part remains unchanged and is disjoint from the part used by the target.

B.2 Multiplication by an Odd Integer

Let a be an odd coefficient that remains unchanged, and let the target have m bits. Write $z = z_0 + 2^k z_1$ with $k = \lfloor m/2 \rfloor$. First recursively multiply the high half by a modulo 2^{m-k}. Then add $\lfloor az_0/2^k \rfloor \bmod 2^{m-k}$ to that high half, generating its digits from the unchanged low half and the coefficient. Finally recursively multiply the low half by a modulo 2^k. The identity

$$az = (az_0 \bmod 2^k) + 2^k\big(az_1 + \lfloor az_0/2^k \rfloor\big)$$

proves correctness modulo 2^m.

At $m = O(w)$, use a temporary buffer and the multiplication algorithm of Lemma 2. Compute the product into the buffer, swap, and uncompute the old input with the inverse coefficient modulo 2^m, obtained from (58). This uses $O(w)$ ancilla qubits. Above scale w there are $O(n/w) = O(g)$ recursive nodes, each querying at most $O(g)$ digits. The total is $\widetilde{O}(n)$ Toffoli gates with $o(n)$ ancilla qubits. For the inverse multiplication, reverse the sequence of updates and use fresh oracle calls, including their phase corrections.

B.3 Replacement under a Linear Relation

Let the nonzero signed coefficients A, B and constant C remain unchanged. We construct a permutation of all m-bit target strings that maps x to y whenever the integer relation $\mathsf{A}x + \mathsf{B}y = \mathsf{C}$ holds with $0 \le x, y < 2^m$. For a nonzero integer a, let $\nu_2(a)$ be the largest integer $k \ge 0$ such that 2^k divides a. Set $\nu = \min\{\nu_2(\mathsf{A}), \nu_2(\mathsf{B})\}$. If 2^ν does not divide C, there are no integer solutions and we use the identity permutation. Otherwise divide $\mathsf{A}, \mathsf{B}, \mathsf{C}$ by 2^ν; at least one reduced coefficient is odd. In the following, $\mathsf{A}, \mathsf{B}, \mathsf{C}$ denote these reduced coefficients. Their required digits are evaluated from the unchanged parameters.

Both coefficients odd. The affine permutation

$$x \leftarrow \mathsf{B}^{-1}(\mathsf{C} - \mathsf{A}x) \bmod 2^m \tag{61}$$

has output y on inputs satisfying the integer relation. Its slope is odd, so it is a permutation of all m-bit target strings.

Only B *odd.* Write $\mathsf{A} = 2^\alpha a$ with a odd. If $0 < \alpha < m$, set

$$y_0 = \mathsf{C}\mathsf{B}^{-1} \bmod 2^\alpha, \qquad D_0 = (\mathsf{C} - \mathsf{B}y_0)/2^\alpha. \tag{62}$$

Apply the odd affine map

$$x \leftarrow \mathsf{B}^{-1}(D_0 - ax) \bmod 2^m.$$

On inputs satisfying the integer relation, this is $(y - y_0)/2^\alpha$, with its high α bits zero. Rotate the same target register left by α and XOR y_0 into the low bits. This gives y. Both operations are permutations on every target string. Figure 6 shows why the rotation needs no extra target bits.

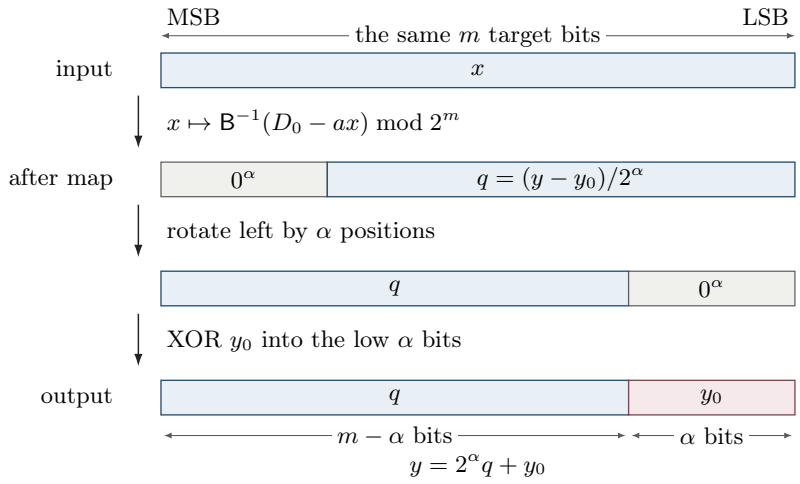

Fig. 6. In-place replacement under $\mathsf{A}x + \mathsf{B}y = \mathsf{C}$ when B is odd and $\mathsf{A} = 2^\alpha a$, with a odd and $0 < \alpha < m$. Here $y_0 = \mathsf{C}\mathsf{B}^{-1} \bmod 2^\alpha$ and $D_0 = (\mathsf{C} - \mathsf{B}y_0)/2^\alpha$, as in (62). On inputs satisfying the integer relation, the affine map produces $q = (y - y_0)/2^\alpha$ with α leading zeros. Rotation moves those zeros to the low end; XOR inserts y_0. The parts holding q and the zeros keep their widths, and every row uses the same m target bits. The displayed zeros are guaranteed only on the stated inputs; every operation is a permutation on all target strings.

Only A *odd.* Write $\mathsf{B} = 2^\beta b_0$ with b_0 odd. If $0 < \beta < m$, set

$$x_0 = \mathsf{C}\mathsf{A}^{-1} \bmod 2^\beta, \qquad D_0 = (\mathsf{C} - \mathsf{A}x_0)/2^\beta. \tag{63}$$

XOR the known low bits x_0 from the target and rotate right by β. On inputs satisfying the integer relation, this leaves $(x - x_0)/2^\beta$ with high bits zero. Then apply

$$x \leftarrow b_0^{-1}(D_0 - \mathsf{A}x) \bmod 2^m$$

to obtain y.

Large valuations. If $\alpha \geq m$ or $\beta \geq m$, the congruence fixes one of x, y modulo 2^m, hence fixes it as a canonical integer. The exact equation determines the other. There is at most one pair satisfying the integer relation and the stated bounds. Evaluate that pair from the unchanged parameters and XOR its known binary difference into the target. If there is no such pair, use the identity permutation.

Each affine map above uses multiplication by an odd integer and addition one digit at a time. Constants such as D_0, x_0, y_0 are generated one digit at a time. Valuations and the controls selecting a case have $O(\log n)$-bit descriptions; a variable cyclic rotation of m cells takes $O(m \log m)$ Toffoli and CNOT gates. All controls selecting a case depend on unchanged parameters and can be uncomputed from them after the target update. Thus every case uses $\widetilde{O}(n)$ Toffoli

gates and $o(n)$ ancilla qubits, without extending the target by the valuation of either coefficient. This proves Lemma 5.

C Selection and Recovery of Small Quotients

We prove Lemma 7, including the treatment of rational endpoint boundaries and recovery from the output after a block of small quotients. The notation and stopping rules are those of Section 4.3.

Proof (Lemma 7). The input satisfies (15) and (19). We first compute the matrix from this input. Test the stopping conditions $r = 1$ and $t \geq U$, followed by $R \geq Br$. If none holds, the current quotient is small. Set

$$\ell_{\mathrm{CF}} = 64b + 256, \qquad A_* = 2^{4b+16}. \tag{64}$$

When $\mathrm{len}(R) \leq \ell_{\mathrm{CF}}$, compute and store the pair (R, r) and use the circuit for ordinary continued fractions on materialized operands. Its input has $O(b)$ bits. Prefix matrices and predicates evaluated on the full EEA state select the first stopping point, or the prefix immediately preceding a large quotient.

Otherwise put $s_{\mathrm{CF}} = \mathrm{len}(R) - \ell_{\mathrm{CF}}$ and compute

$$u = \lfloor R/2^{s_{\mathrm{CF}}} \rfloor, \qquad v = \lfloor r/2^{s_{\mathrm{CF}}} \rfloor, \qquad \mathcal{I} = \left[\frac{u}{v+1}, \frac{u+1}{v} \right].$$

Since $R < Br$, $v > 0$. Moreover $u \geq 2^{\ell_{\mathrm{CF}}-1}$, $v \geq u/(2B)$, and $v \leq u$, so

$$R/r \in \mathcal{I}, \qquad |\mathcal{I}| = \frac{u+v+1}{v(v+1)} < 2^{2b+5-\ell_{\mathrm{CF}}}. \tag{65}$$

Compute the ordinary continued fractions of the two endpoints. An endpoint expansion may begin with a zero quotient when that endpoint is below 1; such a zero is used only to compare the endpoint expansions and is never accepted as a quotient of the actual ratio $R/r > 1$. Keep their common positive-quotient prefix, stopping before a terminating endpoint quotient, before a quotient at least B, or at a disagreement. Also stop as soon as the leading matrix entry reaches A_*. All endpoint data and matrices have $O(b)$ bits; the first matrix whose leading entry reaches the threshold has leading entry less than $(B+1)A_*$.

For forward selection, truncate every candidate quotient list before the Euclidean division that would produce remainder zero; its preceding state has $r = 1$ and is the terminal forward state. Every retained prefix therefore has positive smaller remainder. Every accepted common quotient is a quotient of the actual ratio R/r. For the resulting list of prefix matrices, test the stopping predicate $[r = 1] \vee [t \geq U]$ on the state described by (21). This predicate is monotone along the truncated list, so binary search locates its first true prefix. If the leading entry reaches A_*, the desired stop has already occurred, by (22).

It remains to resolve an endpoint boundary when the stopping predicate has not yet occurred. Let

$$F_M(y) = \frac{ay + b_1}{cy + d}$$

be the map for the retained common prefix. At this point its entries are below A_*. If the actual next quotient is large, a test on the full EEA state ends the block. If the actual remaining ratio is an integer, coprimality forces its smaller remainder to be 1, so forward selection has already terminated. Otherwise write the actual remaining ratio as $y = q + 1/z$ with finite z and $q < B$. Endpoint disagreement, termination, or an endpoint quotient reaching B implies that some adjacent integer $k \in \{q, q+1\}$ has $F_M(k) \in \mathcal{I}$. Indeed the transformed endpoint interval meets a boundary of the continued-fraction cell containing y; the integer endpoints of that cell are q and $q+1$.

Suppose three further divisions with small quotients can be taken before the required stop. Then $z < B+1$. For $k = q$, this gives $|y-k| = 1/z > 1/(B+2)$. For $k = q+1$, either $z \geq 2$, giving a distance at least $1/2$, or $z = 1 + 1/v_0$ with $v_0 < B+1$, giving $|y-k| = 1/(v_0+1) > 1/(B+2)$. Thus in either case

$$|y-k| > \frac{1}{B+2}. \tag{66}$$

Using $|\det M| = 1$, $y < B$, and $k \leq B$, we obtain

$$|F_M(y) - F_M(k)| = \frac{|y-k|}{|(cy+d)(ck+d)|} > \frac{1}{A_*^2(B+2)^3}. \tag{67}$$

This contradicts (65), since

$$2^{2b+5-\ell_{\mathrm{CF}}} < \frac{1}{A_*^2(B+2)^3} \qquad (b \geq 1).$$

At most two additional divisions with small quotients are therefore necessary. They are evaluated as constant-depth expressions in the original state and the retained transition matrix. Before each, test whether $r = 1$, whether the coefficient bound has been reached, and whether the quotient is large. The procedure returns the preceding prefix if a large quotient is encountered.

Recovery from the output. By (19), the block was empty exactly when $T' < L$. If it was nonempty, the last division has $T' = t_{\text{previous}} \geq L$. Starting from (t', T'), run ordinary Euclid backwards through the coefficient recurrence (23) stopping at the first smaller coefficient below L. Its quotients are $q_k, \ldots, q_1$, and their product matrix is M^{T}. The strict coefficient inequalities in (15) determine each quotient uniquely. At $T = 0$, the normalization condition gives $q_1 \geq 2$, so the last reverse quotient agrees with the ordinary terminating convention. All of these quotients are small. The same procedure applies to the enclosing interval, with stopping predicate "smaller coefficient $< L$". If the leading entry of the matrix for the recovered quotients reaches A_*, the remaining larger coefficient is at most $t'/A_* < L$, so this stopping point has already occurred. The same boundary-separation argument applies. If the current reverse ratio is an integer and the stopping predicate is still false, one terminal division gives smaller coefficient zero and ends recovery. Thus an integer ratio requires no further boundary analysis.

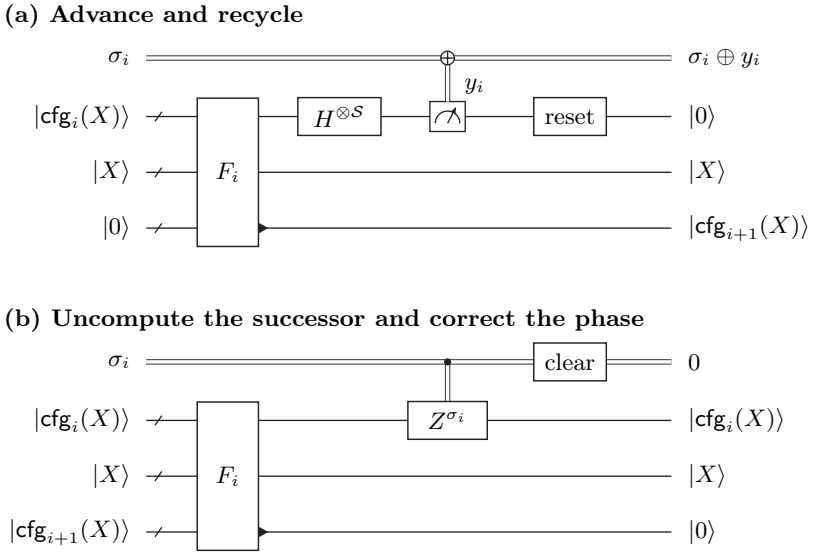

Fig. 7. Two steps of measurement-based uncomputation. The same XOR map for a segment F_i writes the successor in (a) and uncomputes it in (b). Measurement contributes $(-1)^{y_i \cdot \mathsf{cfg}_i(X)}$ and updates the classical record. Before (b), $\mathsf{cfg}_i(X)$ has been recomputed; Z^{σ_i} cancels its accumulated phase and the record is cleared. The $O(\mathcal{S})$ ancillas holding the segment history of F_i are omitted and return to zero at each call. The unchanged input X is never copied into a checkpoint.

In particular, for $L = 1$ the final zero remainder is accepted and is tested before any further division.

The endpoint circuits and their balanced product trees (Section 2.3) occupy $O(b \log^{c_{\mathrm{CF}}}(n+2))$ wires. Prefix queries are performed by their Boolean circuits, including routing, and cost $b \log^{O(1)}(n+2)$ operations each. There are $O(\log(b+2))$ predicate queries in the binary searches and at most two divisions evaluated on the full EEA state. Each query is a constant-depth expression in the unchanged input subregisters and a materialized prefix matrix, and therefore satisfies Lemma 3. The oblivious scans cover every case. This proves the claimed classical bounds in both directions.

D Measurement-Based Uncomputation

Figure 7 separates two operations from Lemma 4. The XOR map for a segment F_i preserves X and $\mathsf{cfg}_i(X)$ while XORing the next working state into its target. Its $O(\mathcal{S})$ ancillas holding the segment history, omitted from the diagram, return to zero. The two panels are not consecutive: the backward sweep first recomputes $\mathsf{cfg}_i(X)$, possibly creating further measurement records, before using panel (b). The record σ_i is the XOR of all measurements of that working state.

Proof (Lemma 4). Split the program into $Q_{\mathrm{seg}} = \lceil \mathcal{T}/\mathcal{S} \rceil$ segments of at most $\mathcal{S}$ operations, and let $\mathsf{cfg}_i(X)$ be the working configuration after segment i, with cfg_0 the input-independent initial configuration. A segment map F_i XORs $\mathsf{cfg}_{i+1}(X)$ into a target given X and $\mathsf{cfg}_i(X)$: copy cfg_i with CNOTs, execute the segment keeping every overwritten bit, copy the result into the target, and run the segment and the copy backwards. This costs $O(\mathcal{S})$ Toffoli gates and $O(\mathcal{S})$ ancillas, and reads X in place.

Forward. Compute $\mathsf{cfg}_1, \mathsf{cfg}_2, \ldots, \mathsf{cfg}_{Q_{\mathrm{seg}}}$ in turn with a constant number of configuration buffers. Whenever a buffer holding cfg_i is no longer needed, measure it in the Hadamard basis, obtaining y_i, reset it, and XOR y_i into a classical record σ_i; by (8) the state acquires $(-1)^{y_i \cdot \mathsf{cfg}_i(X)}$. Copy the output into the target register.

Backward. Maintain the invariant that at stage j only $\mathsf{cfg}_j(X)$ is live, all records σ_i with $i \geq j$ are zero, and the accumulated phase is $(-1)^{\sum_{i<j} \sigma_i \cdot \mathsf{cfg}_i(X)}$ up to an X-independent normalization. Recompute cfg_{j-1} from cfg_0, measuring away the intermediate buffers as before, which adds to records with index below $j-1$ only. Use F_{j-1} to erase cfg_j, apply Z to the positions of the live cfg_{j-1} where σ_{j-1} is one, which multiplies by $(-1)^{\sigma_{j-1} \cdot \mathsf{cfg}_{j-1}(X)}$, and clear the working record σ_{j-1}. The complete outcome record μ is still retained in the branch description. The invariant now holds at $j-1$. At $j = 0$ no phase remains and cfg_0 is cleared.

Cost and exactness. There are $O(Q_{\mathrm{seg}}^2)$ segment executions, each $O(\mathcal{S})$ Toffoli gates, so $O(Q_{\mathrm{seg}}^2 \mathcal{S}) = O(\mathcal{T}^2/\mathcal{S} + \mathcal{T})$ gates, with $O(\mathcal{S})$ ancillas. The positions of all measurements are chosen in advance. Each outcome is uniform and its phase is a character of a function of X that the correction removes. If M_{meas} qubits are measured over the complete input-independent schedule, the Kraus operator for that measurement record is $2^{-M_{\mathrm{meas}}/2} O_f$ on the prescribed zero-workspace subspace. This scalar is independent of X, and linearity covers inputs entangled with a reference.

E Phase Estimation and Postprocessing

The details below complete the ECDLP reduction of Section 6: first the semiclassical phase-estimation schedule using one reusable control qubit, then the classical recovery of the subgroup order and discrete logarithm from two joint samples.

E.1 Semiclassical Phase Estimation

For $A \in \{G, Q\}$, consider an eigenstate of addition by A with eigenvalue $e^{2\pi i \theta}$. In round $j = m-1, \ldots, 0$, prepare the reset control in $|+\rangle = H|0\rangle$, apply the controlled addition of $2^j A$, followed by

$$R_\mu^{(j)} = \mathrm{diag}\left(1, e^{-2\pi i \rho_j}\right), \qquad \rho_j = \sum_{\ell=j+1}^{m-1} \frac{\mu_\ell}{2^{\ell-j+1}}, \tag{68}$$

and then a Hadamard gate and measurement of μ_j. The sum uses previously measured bits and is empty in the first round. The phase estimate is

$$\widehat{\theta} = \sum_{j=0}^{m-1} \frac{\mu_j}{2^{j+1}} = 0.\mu_0\mu_1\cdots\mu_{m-1}.$$

Thus the bits are measured from least to most significant. Figure 8 shows the order of the operations.

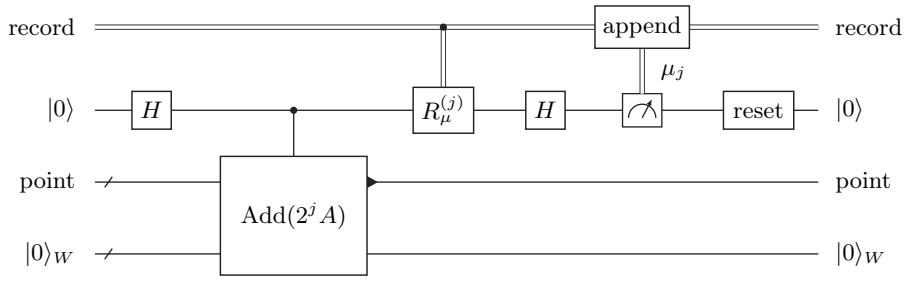

Fig. 8. One round of Lemma 14. Controlled addition of $2^j A$ returns W to zero. The rotation uses prior outcomes; μ_j is measured and appended before reset, and the point register is reused for both estimates of a joint sample.

E.2 Postprocessing Formulas

An attempt uses two independent joint samples, resetting the point register between them. Write $\theta_i = a_i/s_i$ in lowest terms and $\widehat{r} = \operatorname{lcm}(s_1, s_2)$. For correct samples and independent uniform indices,

$$\widehat{r} \mid r_G, \qquad \widehat{r} = r_G \iff \gcd(k_1, k_2, r_G) = 1, \tag{69}$$

$$\Pr[\gcd(k_1, k_2, r_G) = 1] = \prod_{\ell \mid r_G \text{ prime}} (1 - \ell^{-2}) \geq \zeta(2)^{-1} > 0.6. \tag{70}$$

Check $\widehat{r}G = \mathcal{O}$, integrality of $k_i = \widehat{r}\theta_i$ and $l_i = \widehat{r}\phi_i$, and $\gcd(k_1, k_2, \widehat{r}) = 1$. Choose u_1, u_2, v with $u_1k_1 + u_2k_2 + v\widehat{r} = 1$, set $d = u_1l_1 + u_2l_2 \pmod{\widehat{r}}$, and accept only if $dG = Q$. Correct samples with coprime indices give $\widehat{r} = r_G$ and $l_i \equiv k_i d \pmod{r_G}$, hence the right d.