Quantum Resource Optimization for CSIDH

Yan Huang[1], Yongjie Li[1], Xiuyu Qiu[5], Zijian Zhou[3], Fangguo Zhang[2], Chao Chen[4], and Wei Yu[5⋆]

[1] School of Mathematics and Statistics, Hunan University of Science and Technology, Xiangtan 411201, China.
[2] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China.
[3] College of Science, National University of Defense Technology, Changsha 410083, China.
[4] State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100085, China.
[5] College of Big Data and Statistics, Sichuan Tourism University, Chengdu 610100, Sichuan, China.

Abstract. The quantum resource analysis of CSIDH has remained an active research topic. At Eurocrypt 2020, Peikert raised three open problems concerning the optimization of the corresponding quantum resources. In this work, we primarily focus on the first two: optimizing quantum resources for ideal class groups and choosing the collimation arity. Specifically, we reduce the T-gate complexity of class group actions for CSIDH-512 from $2^{52.6}$ to $2^{51.7}$. Furthermore, within our four-way permutation construction model and under a specified classical memory budget, we provide a detailed analysis of the classical and quantum resources associated with different collimation arities r. Among the evaluated configurations, $r = 4$ emerges as the optimal choice. By incorporating the hidden-shift quantum algorithms proposed by Peikert, we achieve a T-gate reduction of at least 85% for solving CSIDH-512.

Keywords: Isogeny-based cryptography · CSIDH · Quantum circuits

1 Introduction

CSIDH instantiates a commutative group action from ideal classes of an imaginary quadratic order acting on supersingular elliptic curves over a prime field [1]. Its most detailed quantum security analyses combine a hidden-shift algorithm with a quantum circuit that evaluates the class-group action in superposition [2][3].

Castryck, Lange, Martindale, Panny and Renes proposed CSIDH as a candidate post-quantum "commutative group action", which attracted much attention and interest, due to its small communication and relatively high efficiency [1]. Authors suggested concrete parameters in order to meet some desired levels of quantum security. Bonnetain and Schrottenloher conducted a detailed quantum

⋆ Corresponding author, email: yuwei@iie.ac.cn

security analysis of CSIDH [2]. This mainly includes two parts: first, a quantum algorithm for recovering a hidden shift in a commutative group. Second, a computation in superposition of all isogenies originating from a given curve.

For recovering a hidden shift, Bonnetain and Schrottenloher proposed three quantum algorithms [2]: one using a combination routine on labeled qubits, another based on subset-sums, and the third derived from Kuperberg's quantum algorithm [4]. Among them, the second method required the fewest quantum queries, i.e., $2\log_2(\log_2 N)$ where N represents the size of the class group. Peikert generalized Kuperberg's collimation sieve [4] to work for arbitrary finite cyclic groups, provided some practical efficiency improvements for recovering a hidden shift [3]. Chávez-Saabet al.[5] provided three different permutations for Peikert's collimation sieve: (1) classical traversal/iteration approach, (2) divide-and-conquer merging and (3) Grover's algorithm. When r is small, e.g., $r = 2, 3$, Method (1) is preferable; when $r \geq 5$, Method (3) is preferable, but this is subject to depth constraints. If there is no depth limitation, Method (2) is the best choice.

The second part is to design a quantum circuit for the class group action, which mainly includes quantum algorithms for modular arithmetic operations (e.g., modular multiplication, modular inversion and modular exponentiation), scalar multiplication, as well as isogeny and isogeny-based curves. Bonnetain and Schrottenloher proposed the first corresponding quantum algorithm[2]. For computationally expensive modular multiplication and modular inversion, they adopted the approach of Roetteler et al. [6], which involves converting classical Montgomery-based multiplication and extended Euclidean-based inversion algorithms into their corresponding quantum versions [2]. For modular exponentiation, they designed the corresponding quantum algorithm by combining the classical square-and-multiply technique with Bennett's time-space trade-off method. For isogeny and isogeny curve computations, they developed the corresponding quantum algorithm by adapting the classical algorithm from [7] and incorporating Bennett's time-space trade-off method.

Peikert raised three open problems concerning the optimization of quantum resources: (1) the optimization of quantum oracles, namely the quantum resource optimization for evaluating the CSIDH group action; (2) the choice of the collimation arity r; and (3) the number of bits of the CSIDH secret key that must be recovered to compromise the scheme [3]. Our contributions are primarily centered on the former two, which are detailed as follows.

Fixed-exponent Legendre-symbol computation. To compute modular exponentiation of the form

$$x^{\frac{(p-1)}{2}}, \quad x \in \mathbb{F}_p, \tag{1}$$

We propose an efficient reversible quantum algorithm for fixed-exponent Legendre-symbol computation. By combining addition-chain windowing, precomputation tables, Bennett–Knill pebbling, and inverse-operation uncomputation, the algorithm achieves a significant 60% reduction in T-gate count for the CSIDH-512 setting. This work provides a critical building block for the overall optimization of quantum circuits for class-group actions.

Isogeny computaions. For isogeny computations, we build on the observation that classical implementations on Edwards curves already attain optimal efficiency, and we accordingly enhance the performance of their quantum counterparts. When applied to CSIDH-512, our empirical results demonstrate that the T-gate count is reduced by at least 4% for the out-of-place isogeny curve.

Four-list collimation permutation. In the collimation permutation problem [5], we reinstate the two-scale setting with parameters $S_1 \gg S_2$. For the arity-four case, the algorithm employs two heap-generated pair-sum streams together with a reusable active window to enumerate all surviving tuples. The algorithm runs in $O(L^2 \log L + |J|)$ time and uses $O(L + W_{\max})$ working words, in addition to the unavoidable output table; under the random-list sieve model, the expected size of the active window is on the order of $L^2 S_2/S_1$. We implement the 4-ary collimation sieves in Haskell based on Peikert's reference implementation of the simulator and confirm that $r = 4$ substantially reduces the recursion depth and oracle-query complexity compared with $r = 2$. For other values of r, our theoretical analysis reveals that as r increases, the total T-gate count exhibits a steady decline, while the classical time and memory overheads associated with the collimation phase grow proportionally. In particular, this trade-off suggests an optimal operating regime for intermediate r, balancing quantum resource savings against classical preprocessing costs.

Efficiency Analysis. Combining the optimization techniques described above, we achieve a reduction in the ideal class group complexity for CSIDH-512 from $2^{52.6}$ to $2^{51.7}$, corresponding to a 85% decrease in the overall complexity of solving the underlying isogeny problem.

2 Preliminaries

2.1 Quantum arithmetic operations over the base field $\mathbb{F}_p$

For modular algebraic operations over $\mathbb{F}_p$ where $n = \lceil \log p \rceil$, the key operations include modular addition, squaring, multiplication, inversion, and division. For modular addition, since integer addition can be classified into three types based on the least significant qubit [8], minimal quantum T-depth[9], or the fewest T gates [10], Häner et al. subsequently derived corresponding modular addition circuits from these perspectives [11]. For modular multiplication, Häner et al. proposed a serial multiplication method, optimizing the Montgomery multiplication previously presented by Roetteler et al. [6]. For squaring operations, Roetteler et al. designed a corresponding quantum circuit based on the double-and-add method [6]. By optimizing the addition operation in terms of minimal qubit usage, lowest quantum depth, and fewest T gates, Häner et al. subsequently proposed an improved squaring implementation [11]. For the inversion operation, Roetteler et al. developed a quantum algorithm based on the extended Euclidean algorithm, later optimized by Häner et al. via SWAP gate reductions.

To compute $x/y \mod p$, Roetteler et al. first inverted y in an auxiliary register, multiplied the result by x, and then uncomputed the auxiliary qubits by inverting y again.

Gu et al. [13] also proposed a newer minimum-T arithmetic family. We retain those rows in Table 1. These circuits use a different architecture and uncomputation model, including a two-dimensional nearest-neighbor layout and dynamic-circuit ingredients. They are therefore informative alternatives, but they are not substituted component-by-component into the Bonnetain–Schrottenloher oracle. The concrete comparison in Section 5 uses one homogeneous model throughout, i.e.

$$n = 511, \qquad T_M = 4n^2 = 2^{19.99} \text{ Toffoli gates}, \qquad Q_M = 3n, \tag{2}$$

and

$$T_I = 32n^2 \log_2 n = 2^{26.16} \text{Toffoli gates}, \qquad Q_I = 5n + 2\lceil \log_2 n \rceil + 7. \tag{3}$$

Table 1: Restored alternative base-arithmetic estimates from [13]; these rows are not mixed into the homogeneous CSIDH-512 comparison.

Circuit	Depth	T gates	Logical qubits
Addition	$4n - 2$	$16n + 4$	$4n + 4$
Squaring	$21.4n^2 + 1.43 \cdot 2^{10}$	$37.8n^2 + 1.4 \cdot 2^{12}$	$7n + 12.2$
Multiplication	$21.4n^2 + 1.35 \cdot 2^{10}$	$37.8n^2 + 1.4 \cdot 2^{12}$	$7n + 12.2$
Inversion	$23.14n^2 + 1.25 \cdot 2^9$	$70.85n^2 + 1.94 \cdot 2^{11}$	$10n + \lfloor \log_2 n \rfloor + 11$
Division	$26.2n^2 \log_2 n + 1.67 \cdot 2^9$	$76.29n^2 \log_2 n + 1.18 \cdot 2^{18}$	$10n + \lfloor \log_2 n \rfloor + 11$

We concisely denote these quantum algorithms as $add\_modp, squ\_modp, mul\_modp$ and div, while f and g represent quantum states of elements in $\mathbb{F}_p$. Then, these algorithms are simply described as follows:

$$\begin{aligned}
&add_modp : |f\rangle|g\rangle \to |f\rangle|f + g \mod p\rangle;\\
&squ_modp : |f\rangle|0\rangle \to |f\rangle|f^2 \mod p\rangle;\\
&mul_modp : |f\rangle|g\rangle|0\rangle \to |f\rangle|g\rangle|fg \mod p\rangle;\\
&div_modp : |f\rangle|g\rangle|0\rangle \to |f\rangle|g\rangle|fg^{-1} \mod p\rangle.
\end{aligned}$$

2.2 Collimation Sieve

Kuperberg proposed an algorithm for solving the hidden-shift problem, consisting of two main components: (1) a quantum oracle that evaluates the group action on a uniform superposition and produces random phase vectors;(2)a sieving procedure that destructively combines low-quality phase vectors into high-quality phase vectors [4]. Peikert has proposed a sieve method for collimating on the most-significant bits, built upon Kuperberg's algorithm and leveraging quantum accessible classical memory (QRACM) in Algorithm 1[3]. U_f is a quantum

oracle that maps $|x\rangle|0\rangle$ to $|x\rangle|f(x)\rangle$. The phase vector $|\varphi_i\rangle$ (for $i = 1, 2, \ldots, r$) has length L_i and is defined as

$$|\varphi_i\rangle = L^{-1/2} \sum_{j \in [L_i]} \chi(b_i(j)s/N)|j\rangle,$$

$b_i(j)$ is a map from $[L_i]$ to $\mathbb{Z}$.

Algorithm 1 Collimation sieve for group $\mathbb{Z}_N$ and collimation arity r

Input: Interval sizes $S_0 < S_1 < \cdots < S_d = N$, a desired phase-vector length L, and oracle access to U_f.
Output: A phase vector on $[S_0]$ of length $\approx L$.
Base case. If $S_0 = N$, generate $\ell \approx \log L$ length-2 phase vectors $|\varphi_1\rangle, |\varphi_2\rangle, \ldots, |\varphi_r\rangle$ using U_f. Output the length-2^ℓ phase vector $|\varphi\rangle = |\varphi_1, \ldots, \varphi_r\rangle$.
Recursive case. Otherwise:

1: Using r recursive calls for sizes S_{i+1} and appropriate desired length, obtain r phase vectors $|\varphi_1\rangle, |\varphi_2\rangle, \ldots, |\varphi_r\rangle$ on S_i;
2: Form the phase vector $|\varphi'\rangle = |\varphi_1, \varphi_2, \ldots, \varphi_r\rangle$ having index set $[L_1] \times \cdots \times [L_r]$ and phase multiplier function $b'(j) = \sum_{i=1}^{r} b_i(j_i)$;
3: Measure $|\varphi'\rangle$ according to the value of $k = \left\lfloor \frac{b'(j)}{S_i} \right\rfloor$ to obtain $P_k |\varphi'\rangle$ for a certain subunitary P_k;
4: Find the set J of the tuples j that satisfy the above. Let $L = |J|$ and choose a bijection $\pi : J \to [L]$;
5: Output phase vector $|\varphi\rangle = U_\pi P_k |\varphi'\rangle$ with index set $[L]$ and multiplier function $b(j) = b'(\pi^{-1}(j))$.

Recursion Depth. The recursion depth d of the collimation sieve is determined by r and L. The sequence of interval sizes S_i satisfies:

$$S_0 \approx L, S_{i+1} = \min\left\{\approx S_i \cdot L^{r-1}/r, N\right\},$$

where $S_d = N$ is the length of the initial phase vectors, and the recursion terminates when $S_i = S_0 = L$. Solving the recurrence yields the condition :

$$S_0 \left(L^{r-1}/r\right)^d \geq N \quad \Longrightarrow \quad d = \left\lceil \frac{\log(N/S_0)}{\log(L^{r-1}/r)} \right\rceil.$$

Query Complexity. At the leaf level $S_d = N$, the phase vector length is approximately:

$$L' = \left(rLN/S_{d-1}\right)^{1/r}.$$

This requires $\lceil \log L' \rceil$ oracle queries. Accounting for discards with fraction δ, the effective branching factor is $r/(1-\delta)$. Total oracle queries:

$$Q = \left(r/(1-\delta)\right)^d \cdot \log L'.$$

To recover the secret key, set $S = L$. Each sieve run yields an expected $\log S - 2$ bits, and the remaining 56 bits are brute-forced. Therefore, the total number of oracle queries becomes:

$$\tilde{Q}_{\text{total}} = (r/(1-\delta))^d \cdot \log L' \cdot \frac{\log N - 56}{\log S - 2}.$$

QRACM Memory. Instead of storing full multipliers $b_i(j_i)$ (requiring $\log N$ bits each), we store only the high-order bits. Letting L_{max} denote the maximum of the lengths of the input and output phase vectors, the number of bits required is:

$$L_{\max} \cdot (1+\alpha) \log\big(L^{r-1}/r\big).$$

The other is the π table employed in Step 4 of Algorithm 1, for which the number of qubits required is:

$$L_{\max} \log L_{\max}.$$

Both types of data reuse the same QRACM region, so the total memory is the larger of the two:

$$R = L_{\max} \cdot \max\left\{(1+\alpha)\log\left(L^{r-1}/r\right), \log L_{\max}\right\}.$$

3 Improved Quantum Circuit for the Class Group Action

In this section, we optimize the cost of the quantum circuit that evaluates the CSIDH group action on a given Montgomery curve E_A represented by $A \in \mathbb{F}_p$:

$$|e_1, ..., e_u\rangle |A\rangle |0\rangle \to |e_1, ...e_u\rangle |A\rangle |L_{l_1}^{e_1} \circ \cdots \circ L_{l_u}^{e_u}(A)\rangle$$

where $L_{\ell_i}^{e_i}$ corresponds to applying $[\mathfrak{l}_i]$ to a given curve for e_i times when $e_i > 0$ or its conjugate $[\overline{\mathfrak{l}_i}]$ to a given curve for $|e_i|$ times when $e_i < 0$.

Bonnetain et al. took an ℓ-isogeny as an example [2] to introduce the detailed quantum algorithm. For an ℓ-isogeny, the procedure generally consists of the following steps:

1). Input $|A\rangle$, and create a superposition of good points of order $p+1$ on E_A;
2). Input $|A\rangle|P\rangle$, and compute $Q = ((p+1))/\ell$ using a reversible Montgomery ladder algorithm;
3). Input $|A\rangle|Q\rangle$, and compute the isogenous curve $B = L_\ell(A)$;
4). Use Bennett's algorithm to uncompute the point Q;
5). Use Bennett's algorithm to uncompute the superposition of good points.

For Step 1), it involves determining whether a point lies on the elliptic curve E_A and whether its order is $p+1$, requiring algorithms for fixed-exponent Legendre-symbol computation and the Montgomery ladder algorithm; Step 2) uses the Montgomery ladder algorithm to obtain a point of order ℓ; For Step 1) and 2), they costs $30B(n,s)T_M+6B(n,4s)T_M$ operations using $Q_M+(4s+3)n$ qubits [2]. Step 3) uses the isogenous curve formula to compute the isogenous curve, it requires a cost of $7B(\frac{\ell-1}{2}+1,s)T_M+6B(\lceil\log_2\ell\rceil,4s)T_M+(4\ell-1)T_I+(4\ell+3)T_M$ operations using $Q_I+(4s+11)n$ qubits according to Bonnetain's conclusion. Steps 4) and 5) use Bennett's algorithm to uncompute the superposition states of the original stored points and the good points, so that the register can be reused later. Among the above five steps, the main operations involve fixed-exponent Legendre-symbol computations, inversion, multiplication, the Montgomery ladder algorithm, and quantum algorithms for isogenies.

Based on the advances made by Gu, Ye, Chen, et al. in the implementation of modular multiplication, squaring, inversion, and division [13], we proceed to optimize the quantum algorithms of fixed-exponent Legendre-symbol computation and isogeny computation.

Remark 1. Notice that the notations T_M, T_S, T_I, T_D and Q_M, Q_S, Q_I, Q_D denote the number of T-gates and qubits for multiplication, inversion and division, respectively.

3.1 Fixed-exponent Legendre-symbol computation

The Legendre-symbol of x modulo p is

$$x^{(p-1)/2} \pmod p = \begin{cases} 0, & x = 0, \\ 1, & x \in (\mathbb{F}_p^\times)^2, \\ -1, & x \in \mathbb{F}_p^\times \setminus (\mathbb{F}_p^\times)^2. \end{cases} \tag{4}$$

This is a quadratic-character computation used in good-point preparation. We use clean, out-of-place primitives—specifically, *squ_mod* and *mul_modp*—along with their exact inverses for uncomputation.

The CSIDH-512 prime is

$$p = 4 \cdot 587 \prod_{\substack{\ell \text{ prime} \\ 3 \le \ell \le 373}} \ell - 1. \tag{5}$$

It has 511 bits; the fixed-exponent is $\frac{p-1}{2}$ which has has 510 bits. For a left-to-right window of width w, precompute the odd powers $x, x^3, \ldots, x^{2^w-1}$. If the certified windows are $a_0; (g_1, a_1), \ldots, (g_m, a_m)$, the accumulator recurrence is

$$y_j = y_{j-1}^{2^{g_j}} x^{a_j}, \tag{6}$$

where $y_0 = x^{a_0}$. The squarings must precede the multiplication by the next window.

Algorithm 2 Full-storage clean evaluation of $x^{\frac{p-1}{2}}$

Input: $x \in \mathbb{F}_p$, public width w, and certified schedule $a_0; (g_j, a_j)_{j=1}^{m}$

Output: $|x\rangle|x^{\frac{p-1}{2}}\rangle$ and all work registers reset

```
Compute x^2
for odd a = 3, 5, ..., 2^w − 1 do
    Compute x^a = x^{a−2} x^2 in a fresh register
end for
Copy x^{a_0} into the first accumulator register
for j = 1 to m do
    Apply g_j out-of-place squarings to fresh accumulator registers
    Multiply by the stored odd power x^{a_j}
end for
Copy the final accumulator to the output
Apply every multiplication and squaring inverse in reverse topological order
return |x⟩ |x^{(p−1)/2}⟩
```

Table 2: Certified forward operation counts for fixed-exponent $(p-1)/2$.

w	Squarings	Multiplications	Total primitives
2	509	170	679
3	509	131	640
4	509	111	620
5	506	99	**605**
6	506	101	607
7	504	126	630
8	503	184	687

Table 2 gives the independently verified forward counts. The precomputation costs one squaring and $2^{w-1}-1$ multiplications.

For $w = 5$, the odd-power table uses one squaring and 15 multiplications, and the main chain uses 505 squarings and 84 multiplications. Thus the forward straight-line program costs

$$506T_S + 99T_M,$$

and the full-storage compute–copy–uncompute circuit costs

$$1012T_S + 198T_M.$$

It uses 608 field registers including input and output.

Proposition 1 (Schedule correctness). *Suppose the certificate verifies the Horner identity*

$$\frac{p-1}{2} = \big(\cdots((a_0 2^{g_1} + a_1)2^{g_2} + a_2)\cdots\big)2^{g_m} + a_m, \tag{7}$$

with every a_j odd and $1 \le a_j < 2^5$. Then Algorithm 2 implements $|x\rangle|0\rangle \mapsto |x\rangle\left|x^{\frac{p-1}{2}}\right\rangle$ for all $x \in \mathbb{F}_p$ and coherently on superpositions.

Proof. Induction in Eq. (6) shows that the final accumulator is $x^{\frac{p-1}{2}}$. The output is copied by CNOTs in the computational-basis encoding. Applying the exact inverse of every primitive in reverse topological order clears all table and accumulator registers without touching the copied output. Equation (4) gives the desired character value, including $x = 0$.

Memory-matched reversible pebbling. The full-storage circuit gives the smallest primitive count in our comparison, but it uses more qubits than the published character circuit. To obtain an apples-to-apples row, we apply the same Bennett–Knill recurrence[14]. Let $B(t, s)$ be the least number of primitive invocations required to reversibly evaluate a chain of t maps with s checkpoint registers, where the input is retained and all temporary checkpoints are cleared. The recurrence is

$$B(1, s) = 1, \qquad B(t, 0) = \infty \ (t \geq 2), \tag{8}$$

$$B(t, s) = \min_{1 \leq k < t} \big(B(k, s) + B(k, s - 1) + B(t - k, s - 1) \big). \tag{9}$$

For width 5, retain 16 additional field registers: one for x^2 and 15 for the odd powers $x^3, x^5, \ldots, x^{31}$. The main accumulator has 589 arithmetic maps. We append one phase-marking map, conservatively charging it as one field primitive, and allocate 44 checkpoint registers. Each out-of-place primitive also needs one field-sized output register in addition to its arithmetic workspace. Exact dynamic programming gives

$$B(590, 44) = 2269. \tag{10}$$

Computing and later erasing the odd-power table costs $2 \cdot 16$ further primitives. Therefore one clean character phase costs at most

$$(32 + 2269)T_M = 2301T_M.$$

The associated local field-register budget is

$$61 = 16 + 44 + 1$$

registers: 16 retained precomputation values, 44 Bennett checkpoints, and one output register for the active out-of-place primitive, plus the arithmetic workspace Q_M and $O(1)$ phase/control bits. 16 retained precomputation values, 44 Bennett checkpoints, and one output register for the active out-of-place primitive, plus the arithmetic workspace Q_M and $O(1)$ phase/control bits.

The pebbling certificate has core SHA-256

`e54389296367bac03e4d6df7a3dc4036ea2eee32d904c57ebea6107e00b4e1fc`.

Its verifier recomputes the recurrence, checks $B(590, 44) = 2269$, imports the independently verified fixed-exponent counts, and regenerates every numerical row in Section 5. The fixed-exponent schedule certificate has core SHA-256

`afbed46943b43f917bc7d8fffbbfa40cb2bceafbc7d1090dde70aa506d2b2d0f`.

3.2 Computing an Isogeny

In this subsection, we optimize the quantum algorithm for Step 3) (isogenous curve computation). Kim et al. found that the w coordinates on Edwards curve had a high efficiency in computing ℓ-isogenous curve [15][16]. According to the formulas of evaluating isogenous curve,

$$d' = d^{\ell}(\prod \frac{w_i + 1}{2})^8$$

Based on this formula, we can readily derive the corresponding quantum algorithm and analyze its quantum resources in Lemma 1.

Lemma 1. ***Out-of-place isogeny curve.*** *There is a circuit that on input $|d\rangle|Q\rangle|0\rangle$, computes $|d\rangle|Q\rangle|d'\rangle$ using $(4s+5)n + Q_D$ qubits.*

$$7B(\frac{(\ell-1)}{2}, s)T_M + 3B(\lceil \log_2 \ell \rceil, 4s)T_M + (\ell-3)(4T_M + 4T_I) + 7T_M + T_D$$

where s denotes a tradeoff parameter, and $B(n,s)$ refers to the function as defined in [18].

Proof. The quantum resource of computing an ℓ-isogenous curve requires analyzing the quantum resources of the following operations.

1. Compute the $w_i = \frac{W_i}{Z_i} (i = 1, 2, .. \frac{(\ell-1)}{2})$ coordinates of points $Q, 2Q, ..., \frac{(\ell-1)}{2}Q$;
2. Compute $h = \prod \frac{w_i+1}{2} = \prod \frac{W_i+Z_i}{2Z_i} = \frac{\prod W_i+Z_i}{\prod 2Z_i}$;
3. Compute h^8;
4. Compute d^{ℓ};
5. Compute $d^{\ell} \cdot h^8$.

For Step 1, according to [2], since the computational cost of point addition and doubling in w-coordinates on Edwards curve is identical to that on Montgomery curves, Step 1 requires $7B(\frac{(\ell-1)}{2}, s)T_M$ Toffoli gates using $n + Q_D$ auxiliary qubits. For Step 2, we employ an in-place multiplication, which requires $2T_M + 2T_I$ Toffoli gates, the computation of h necessitates $2(\ell-3)$ in-place multiplications, thus costs $(\ell-3)(4T_M + 4T_I) + T_D$ Toffoli gates using $Q_D + n$ auxiliary qubits. For Step 3, it involves 5 squaring operations, consuming 5 T_M Toffoli gates while requiring $2n + Q_M$ additional qubits. For Step 4, Bonnetain's method would requires $3B(\lceil \log_2 \ell \rceil, 4s)T_M$ Toffoli gates and $Q_M + (4s+1)n$ ancilla qubits (we can alternatively apply the approach from Algorithm 2 to optimize this computation). For Step 5, computing $d^{\ell} \cdot h^8$ needs two multiplications. Therefore, the total number of Toffoli is

$$7B(\frac{(\ell-1)}{2}, s)T_M + (\ell-3)(4T_M + 4T_I) + 3B(\lceil \log_2 \ell \rceil, 4s)T_M + 7T_M + T_D.$$

For the total number of ancilla qubits, since h and d are computed in the middle, before the backwards recomputation of the ladder steps, this increased the ancilla usage. Thus, the total number of ancilla qubits is

$$n + n + 2n + Q_D + (4s+1)n = (4s+5)n + Q_D.$$

Lemma 2. *[2] There exists a quantum procedure that performs an ℓ-isogeny mapping in place:$|d\rangle \mapsto |L_\ell(d)\rangle$ with an overwhelming probability of success and detectable failure using $T_\ell(s)$ T-gates (including computing good points, Lemma 1 and $30B(n,s)T_M$) and $(4s+7)n+Q_D$ qubits.*

Lemma 3. *[2] Let M be the L_1 bound obtained by reducing the lattice of relations. Assume that $M \lll 2^{50}$ and ℓ is the maximal small prime used. Then there exists a quantum circuit for the class group action using $2MT_\ell(s)$ T-gates and $(4s+7)n+Q_D$ qubits, where s is an integer trade-off parameter, with negligible probability of failure.*

4 An output-sensitive four-list collimation permutation

After obtaining the quotient k through measurement in Step 3 of Algorithm 1, how to efficiently find all tuple indices that satisfy the conditions and construct the permutation π on a classical computer remains one of the core factors affecting the classical computational cost of the entire quantum algorithm. Moreover, the number of subsequent QRACM in Step 4 is also one of the factors influencing the required quantum resources. We begin by formalizing a combinatorial problem that arises in the collimation step of Algorithm 1.

A collimation step receives r sorted lists $B_1, \ldots, B_r$, each of length L, whose entries lie in a range of size S_1. After measuring a bucket index K, the classical permutation builder must enumerate

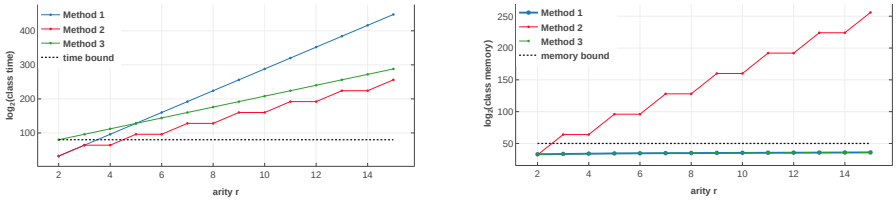
$$J_K = \left\{ (j_1, \ldots, j_r) : KS_2 \le \sum_{i=1}^{r} B_i[j_i] < (K+1)S_2 \right\},$$

where the sieve regime has $S_1 \gg S_2 \gg L$ [5]. For random independent entries, the expected output size is on the scale

$$\mathbb{E}|J_K| \asymp L^r \frac{S_2}{S_1},$$

up to boundary and convolution-density effects. Replacing S_1 and S_2 by one symbol changes this estimate by orders of magnitude and is therefore not a harmless notational simplification.

Three broad approaches are useful for comparison: exhaustive enumeration of $r-1$ lists followed by search in the last, a meet-in-the-middle merge, and quantum search [5]. The plots in Fig. 1 visualize the asymptotic exponents for one representative parameter sweep. For $r > 2$, Method 2 achieves optimal time complexity; however, its memory requirement surpasses the classical memory upper bound. Prompted by this finding, we proceed to optimize Method 2 further. In the following sections, using $r = 4$ as an illustrative example, we introduce a new collimation technique that effectively lowers the classical memory complexity.



Classical-time comparison Classical-memory comparison

Fig. 1: Illustrative asymptotic comparison of candidate collimation-permutation constructions. A concrete arity choice additionally depends on S_1, S_2, output size, depth, QRACM, and hardware limits.

4.1 Heap-generated pair-sum streams

For $r = 4$, define pair sums

$$A = B_1 \oplus B_2, \qquad C = B_3 \oplus B_4,$$

where every item retains both source indices. Because the input lists are sorted, all L^2 items of A can be streamed in increasing order using a min-heap of size L: initialize the heap with $(B_1[i] + B_2[0], i, 0)$, pop the smallest item, and advance only its second index. A symmetric max-heap streams C in decreasing order. Each stream costs $O(L^2 \log L)$ time and $O(L)$ words of working memory without materializing all pair sums.

Let $T_{\min} = KS_2$ and $T_{\max} = (K+1)S_2 - 1$. For a current $a \in A$, the valid values of $c \in C$ lie in

$$I(a) = [T_{\min} - a,\ T_{\max} - a].$$

As a increases, both endpoints of $I(a)$ decrease. We maintain a deque W containing exactly the already generated C items in this interval, in decreasing order. Crucially, items in W are reused for every a they match; they are not destroyed after the first output.

Theorem 1 (Correctness and complexity). *Algorithm 3 outputs every tuple in J_K exactly once. Its time complexity is*

$$O(L^2 \log L + |J_K|),$$

and its working memory, excluding the output table, is

$$O(L + W_{\max}),$$

where $W_{\max}$ is the largest active-window size encountered. Storing the reversible rank table requires an additional $O(|J_K|)$ records, which is unavoidable when the subsequent quantum permutation is implemented by table lookup.

Proof. The heap invariant for A is the standard multiway-merge invariant: the heap contains the least unreported item of each of the L sorted chains $B_1[i] +$

Algorithm 3 Complete four-list interval join

Input: Sorted lists B_1, B_2, B_3, B_4 of length L, scales S_1, S_2, and bucket K
Output: All tuples in J_K and a deterministic rank table

Initialize the increasing heap stream $A = B_1 \oplus B_2$
Initialize the decreasing heap stream $C = B_3 \oplus B_4$
$W \leftarrow$ empty deque; $c \leftarrow$ first item of C; $\rho \leftarrow 0$
for each item $a = (a_0, i_1, i_2)$ from A in increasing order **do**
    lo $\leftarrow KS_2 - a_0$; hi $\leftarrow (K+1)S_2 - 1 - a_0$
    while W is nonempty and W.front.value $>$ hi **do**
        remove the front item of W
    end while
    while c exists and c.value $>$ hi **do**
        advance c in the decreasing stream
    end while
    while c exists and c.value $\geq$ lo **do**
        append c to the back of W; advance c
    end while
    for each (c_0, i_3, i_4) in W **do**
        output (i_1, i_2, i_3, i_4) with rank ρ; $\rho \leftarrow \rho + 1$
    end for
end for

$B_2[0], \ldots, B_1[i] + B_2[L-1]$. Thus every indexed pair appears exactly once and in nondecreasing order. The same argument gives the decreasing C stream.

Fix an iteration with current interval $[\text{lo}, \text{hi}]$. Values above hi are removed from the front of W or discarded from the unconsumed stream; because future intervals only move downward, these values can never become valid later. The second insertion loop consumes every as-yet-unseen C item down to lo and appends it to W. No element below lo is consumed. Hence, after the updates, W contains exactly the indexed C items whose values lie in the current interval. Pairing the current indexed a with every item of W therefore outputs precisely all valid quadruples containing that a, including duplicates of equal numerical sums with different indices. Since each a occurs once, each quadruple occurs once.

The two pair streams perform L^2 heap pops and at most L^2 pushes, each in $O(\log L)$ time and $O(L)$ heap space. Every deque item is inserted and removed at most once, while iterating over active items is charged to an output. This gives the claimed bounds.

4.2 From the classical table to a reversible permutation

Heap ties are broken lexicographically by source indices, and deque iteration follows decreasing pair-sum order with the same tie rule. The resulting generation order in Algorithm 3 is public and deterministic, so the online rank counter defines a bijection

$$\pi_K : J_K \longrightarrow \{0, \ldots, |J_K| - 1\}.$$

The forward and inverse maps are stored in a QROM table and queried coherently. The resource table must charge both query and unquery, the valid-address state preparation when $|J_K|$ is not a power of two, and the table width $4\lceil \log_2 L \rceil$ plus rank metadata. The working memory used to construct the table classically is distinct from the QRACM or QROM memory used by the quantum sieve.

4.3 Experiments and Results

We implement the 4-ary collimation sieves in Haskell based on Peikert's reference implementation of the simulator, which simulates and benchmarks a generalization of Kuperberg's quantum collimation sieve algorithm for arbitrary cyclic groups. The original code provides the sieve where $r = 2$, phase-vector generation, regularization, and numerical Fourier success estimates. We extend the simulator with an $r = 4$ collimation routine that can explicitly construct the inverse table inv and its reverse map π. All experiments reported below use $N = 2^{80}$.

The sieve simulator allows the user to specify:

- a group order N;
- a desired typical phase vector length L;
- an interval size S for the ultimate phase vector;
- the collimation arity $r \in \{2, 4\}$;

The simulator returns its results in a human-readable form together with sieve statistics, including:

- the total number $\tilde{Q}$ of queries to the quantum oracle U_f;
- the number Q of predicted queries;
- the length $\tilde{L}_{\max}$ of the longest created phase vector;
- the probability of obtaining a *regular* phase vector from the final one, and the expected number of bits of the secret that can be recovered from the final phase vector via regularity;
- the probabilities of obtaining *punctured* regular phase vectors of sufficient length from the final phase vector, and the total probability of measuring a value that yields $\log_2 S$secret bits.

Table 3 presents the full-regularization results with $L = 64S$. For the largest parameter set, both arities achieve a regularization probability of approximately 74%, supporting Peikert's observation that $L = 64S$ is generally sufficient to obtain a regular phase vector with substantial probability. The low probability of 34% in one binary run reflects the randomness of the final phase multiplicities, since each row reports one representative execution rather than an average.

Increasing the arity from 2 to 4 reduces the oracle-query count by factors of approximately 8.17, 6.30, and 4.38 for the three parameter sets. The recursion depth decreases from 9, 8, and 7 to 3. Moreover, the 4-ary measurements agree closely with the calibrated query model, with $\widetilde{Q}/Q$ equal to approximately 1.06, 1.00, and 1.03.

$\log_2 N$	r	$\log_2 \widetilde{Q}$	$\log_2 Q$	$\log_2 \widetilde{L}_{\max}$	$\log_2 L$	$\log_2 S$	Pr[regular] (%)	bits	threshold	discard (%)	depth
80	2	12.1	11.7	12.5	10	4	87	3.5	0.25	1.5	9
80	4	9.0	9.0	11.8	10	4	82	3.3	0.25	2.1	3
80	2	11.2	11.0	14.1	11	5	34	1.7	0.25	2.0	8
80	4	8.5	8.5	12.1	11	5	74	3.7	0.25	0.0	3
80	2	10.3	10.3	14.2	12	6	73	4.4	0.25	1.4	7
80	4	8.2	8.2	13.5	12	6	74	4.4	0.25	1.1	3

Table 3: Representative full-regularization runs at $N = 2^{80}$ and $L = 64S$. The column definitions follow Peikert's Fig. 2 [3], with the arity r added. Here N is the group order, r is the collimation arity; $\widetilde{Q}$ and Q are the actual and model-predicted numbers of queries to the quantum oracle; $\widetilde{L}_{\max}$ is the maximum length of all phase vectors created during the run, and L is the requested phase-vector length; S is the range size of the final phase vector. Pr[regular] is the probability of obtaining a regular vector from the final phase vector, and "bits" is the expected number of secret bits recoverable from it. The "threshold" is the minimum permitted ratio between a created vector's length and its requested length; "discard" is the percentage of recursive calls rejected for falling below this threshold; and "depth" is the recursion depth of the sieve. Each row reports one run.

$\log_2 N$	r	$\log_2 \widetilde{Q}$	$\log_2 Q$	$\log_2 \widetilde{L}_{\max}$	$\log_2 L$	$\log_2 S$	p_1	bits	threshold	discard (%)	depth
80	2	16.9	16.7	10.4	6	6	0.302	4.3	0.25	1.9	14
80	4	11.9	11.8	8.7	6	6	0.337	4.4	0.25	1.3	5
80	2	13.8	13.0	11.2	8	8	0.541	7.1	0.25	1.8	10
80	4	10.0	9.7	11.0	8	8	0.393	6.7	0.25	1.3	4
80	2	11.2	11.1	12.3	10	10	0.408	8.7	0.25	1.9	8
80	4	8.6	8.6	11.3	10	10	0.295	8.2	0.25	1.1	3

Table 4: Representative punctured-regularization runs at $N = 2^{80}$ and $L = S$. The format follows Peikert's Figure 3 [3]. Here N is the group order and r is the collimation arity; $\widetilde{Q}$ and Q are the actual and model-predicted numbers of queries to the quantum oracle; $\widetilde{L}_{\max}$ is the maximum length among all phase vectors created during the run, whereas L is the requested phase-vector length; and S is the range size of the final phase vector. The additional quantity p_1 is the probability weight of the first punctured regular vector, obtained by retaining one copy of every distinct multiplier in the final phase vector. Here "bits" is the expected number of secret bits recoverable using the punctured regular vectors derived from the final phase vector. The "threshold" is the minimum permitted ratio between a created vector's length and its requested length; "discard" is the percentage of recursive calls rejected for falling below this threshold; and "depth" is the recursion depth of the sieve. Each row reports one run.

Table 4 reports the punctured-regularization results with $L = S$. The 4-ary sieve reduces the query count by factors of approximately 32.1, 14.0, and 5.87, while reducing the recursion depth from 14, 10, and 8 to 5, 4, and 3, respectively.

The query reduction is obtained at the cost of additional classical computation. As shown in Table 5, the $\widetilde{O}(L^2)$ four-list merge becomes dominant as L increases. In the largest full-regularization run, $r = 4$ uses approximately 4.38 times fewer oracle queries but is much slower than $r = 2$.

Overall, the experiments confirm that $r = 4$ substantially reduces the recursion depth and oracle-query complexity while preserving useful regular or punctured phase vectors. This reduction is obtained at the cost of the greater classical work required by the four-list collimation.

As an additional implementation check, we independently reimplemented the $r = 4$ collimation in Rust, including construction of inv. For the six configurations listed above, in the same order, the values of $\log_2(\text{mean}\,|J_K|)$ over 50 seeded runs were 10.386, 11.141, 12.521, 6.793, 8.307, and 10.186. The corresponding mean wall-clock times for five sequential mapped runs were 0.970, 3.637, 18.535, 0.065, 0.232, and 0.869 seconds.

experiment	$(\log_2 L, \log_2 S)$	$r = 2$ time (s)	$r = 4$ time (s)	$r = 4/r = 2$	$r = 4$ peak residency
full	$(10, 4)$	0.963	6.352	6.6	0.80 MB
full	$(11, 5)$	0.492	25.202	51.2	1.55 MB
full	$(12, 6)$	0.433	159.443	368.2	2.65 MB
punctured	$(6, 6)$	17.293	0.832	0.05	0.18 MB
punctured	$(8, 8)$	1.622	1.022	0.63	0.36 MB
punctured	$(10, 10)$	0.482	5.892	12.2	0.75 MB

Table 5: Measured wall-clock time and maximum live heap of the Haskell implementation. The $r = 2$ and $r = 4$ columns report complete sieve runs under the same experimental configurations. Each timing entry reports one complete representative run.

5 Results and Comparison

In this section, we analyze the quantum resources required for solving the entire CSIDH, and then compare these resources with previous results.

Leveraging the optimizations for modular multiplication, modular inversion, and modular division introduced [13] by Gu et al., we begin by assessing the quantum resources required for fixed-exponent Legendre-symbol computation and isogeny operations in Table 6. The empirical results reveal that the T-gate count drops by at least 60% and 4%, respectively.

We next evaluate the quantum resource costs associated with class group actions, as detailed in Table 7. According to Lemma 2, performing a single in-place isogeny evaluation over a 512-bit prime field requires $197511T_M + 2336T_I + T_D$ T-gates and $(4s + 8)n + Q_D$ qubits. Extending this to the 1300 isogeny evaluations needed for the full protocol, the total T-gate cost becomes $2^{28.9}T_M +$

Basic arithmetic operations	Method in [2]		This work	
	T-gates	Qubits	T-gates	Qubits
Fixed-exponent Legendre-symbol	$5775T_M$	11766	$2301T_M$	11766
Out-of-place isogeny curve	$15116T_M + 2347T_I$	37839	$15057T_M + 2336T_I$	36817

Table 6: Arithmetic estimates [2]

$2^{22.5}T_I + 2^{11.3}T_D$. When incorporating the baseline costs for modular multiplication, inversion, and division—the total T-gate budget for the 512-bit class group action reaches $2^{51.7}$. This marks a 46% improvement in T-count over the prior state-of-the-art result of $2^{52.6}$ T-gates, as reported in [2].

Bit-size n of p	Number M of isogenies	Total T-gates	Ancilla qubits
Method in[2]	1300	$2^{52.6}$	39347[2]
This work	1300	$2^{51.7}$	38861

Table 7: Quantum T-gates and qubits for the class group action for CSIDH-512

	Recursion depth	Total T-gate	QRACM memory	Classic time	Classic memory
$r = 2$	8	$2^{67.7}$	$2^{40.5}$	2^{32}	2^{33}
$r = 3$	4	$2^{65.9}$	$2^{41.5}$	2^{64}	$2^{33.58}$
$r = 4$	3	$2^{65.8}$	2^{42}	2^{64}	2^{34}
$r = 5$	2	$2^{64.2}$	$2^{42.6}$	2^{96}	2^{96}
$r = 6$	2	$2^{64.8}$	$2^{42.9}$	2^{96}	2^{96}
$r = 7$	2	$2^{65.2}$	$2^{43.2}$	2^{128}	2^{128}
$r = 8$	2	$2^{65.6}$	$2^{43.4}$	2^{128}	2^{128}
$r = 9$	1	$2^{62.7}$	$2^{43.6}$	2^{160}	2^{160}
$r = 10$	1	$2^{62.9}$	$2^{43.7}$	2^{160}	2^{160}
$r = 11$	1	2^{63}	$2^{43.9}$	2^{192}	2^{192}
$r = 12$	1	$2^{63.3}$	2^{44}	2^{192}	2^{192}
$r = 13$	1	$2^{63.2}$	$2^{44.2}$	2^{224}	2^{224}
$r = 14$	1	$2^{63.3}$	$2^{44.3}$	2^{224}	2^{224}
$r = 15$	1	$2^{63.4}$	$2^{44.4}$	2^{256}	2^{256}

Table 8: Performance comparison for different r values

Furthermore, extending the four-list merging framework, we evaluate the quantum resource demands for CSIDH-512 across different collimation arities r, as summarized in Table 8. The target length and range size of the sieve's final-stage vectors are both set to $L = S = 2^{32}$, with the maximum observed vector length capped at $\tilde{L}_{\max} = 8L$, and $\alpha = 1/2$. The discard probabilities are specified as $\delta_1 = 0.02$ for $r < 4$ and $\delta_2 = 0.05$ for $r \geq 4$.

Using the CSIDH-512 group order, the figure presents data for r in the range 2 to 15. As r grows, the total T-gate count declines steadily, whereas the classical time and memory overheads of the collimation phase increase proportionally. Thus, selecting r requires a balanced perspective: quantum resource savings must be weighed against the classical cost. The viability of any approach on classical hardware hinges on its asymptotic time and space complexity.

Imposing classical feasibility limits of 2^{80} operations and 2^{40} bytes of memory—representative of practical bounds for classical preprocessing in quantum cryptanalysis—$r = 4$ offers the optimal compromise. For $r > 4$, either the classical time or memory footprint surpasses these thresholds (Table 8). However, should these constraints be relaxed, larger r values could lead to superior quantum performance.

6 Conclusion

In this work, we present optimizations to the quantum resources for solving CSIDH, grounded in three technical innovations: (1)fixed-exponent Legendre-symbol computation based on addition-chain windowing, (2) w-coordinate optimization for isogenous curves in the Edwards curve model, and (3) a systematic exploration of the collimation arity r. In the context of Peikert's collimation sieve [3], the application of these techniques leads to a reduction of about 85% in the total T-gate cost for CSIDH.

Looking ahead, several directions merit further exploration. First, the quantum oracle overhead could be diminished through refinements in finite-field arithmetic and isogeny evaluation. Second, provided that classical resources are not a bottleneck, merging strategies for $r > 4$ could yield further improvements in quantum complexity. Third, whether lattice-based methods can reduce the number of oracle calls using new techniques [19], currently $M = 1300$, remains an intriguing open problem.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (Nos. 12401696, 62272491, 62202475) and Quantum Science and Technology-National Science and Technology Major Project 2026ZD0300100.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin T, Galbraith S (eds), *ASIACRYPT 2018*, Springer LNCS, vol. 11274, pp. 395–427, 2018. https://doi.org/10.1007/978-3-030-03332-3_15
2. Bonnetain, X., Schrottenloher, A.: Quantum Security Analysis of CSIDH. In: Canteaut A, Ishai Y (eds), *EUROCRYPT 2020*, Springer LNCS, vol. 12106, pp. 493–522, 2020. https://doi.org/10.1007/978-3-030-45724-2_17

3. Peikert, C.: He gives C-sieves on the CSIDH. In: Canteaut A, Ishai Y (eds), *EUROCRYPT 2020*, Springer LNCS, vol. 12106, pp. 463–492, 2020. https://doi.org/10.1007/978-3-030-45724-2_16
4. Kuperberg, G.: Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2013)*, vol. 8, 2013. https://doi.org/10.4230/LIPIcs.TQC.2013.20
5. Chávez-Saab, J., Chi-Domínguez, J.-J., Jaques, S., et al.: The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents. *Journal of Cryptographic Engineering*, vol. 12, no. 3, pp. 349–368, 2022. https://doi.org/10.1007/s13389-021-00271-w
6. Roetteler, M., Naehrig, M., Svore, K.-M., Lauter, K.: Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms. In: Takagi T, Peyrin T (eds), *ASIACRYPT 2017*, Springer LNCS, vol. 10625, pp. 241–270, 2017. https://doi.org/10.1007/978-3-319-70697-9_9
7. Bernstein, D.-J., Lange, T., Martindale, C., Panny, L.: Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies. In: Ishai Y, Rijmen V (eds), *EUROCRYPT 2019*, Springer LNCS, vol. 11477, pp. 409–441, 2019. https://doi.org/10.1007/978-3-030-17656-3_15
8. Takahashi, Y., Tani, S., Kunihiro, N.: Quantum addition circuits and unbounded fan-out. *Quantum Information & Computation*, vol. 10, no. 9/10, pp. 872–890, 2010.
9. Draper, T.-G., Kutin, S.-A., Rains, E.-M., et al.: A logarithmic-depth quantum carry-lookahead adder. arXiv preprint quant-ph/0406142, 2004.
10. Cuccaro, S.-A., Draper, T.-G., Kutin, S.-A., et al.: A new quantum ripple-carry addition circuit. arXiv preprint quant-ph/0410184, 2004.
11. Häner, T., Jaques, S., Naehrig, M., Roetteler, M., Soeken, M.: Improved quantum circuits for elliptic curve discrete logarithms. In: *PQC 2020*, Springer LNCS, vol. 12100, pp. 425–444, 2020. https://doi.org/10.1007/978-3-030-44223-1_23
12. Koziel, B., Azarderakhsh, R., Jao, D., Mozaffari-Kermani, M.: On Fast Calculation of Addition Chains for Isogeny-Based Cryptography. In: Chen K, Lin D, Yung M (eds), *Inscrypt 2016*, Springer LNCS, vol. 10143, pp. 323–342, 2016. https://doi.org/10.1007/978-3-319-54705-3_20
13. Gu, Q., Ye, H., Chen, J.-J., Ma, X.-F.: Resource analysis of Shor's elliptic curve algorithm with an improved quantum adder on a two-dimensional lattice. arXiv preprint arXiv:2510.23212, 2025.
14. Knill, E.: An analysis of Bennett's pebble game. CoRR abs/math/9508218, 1995.
15. Kim, S., Yoon, K., Park, Y.-H., Hong, S.: Optimized method for computing odd-degree isogenies on Edwards curves. In: Galbraith SD, Moriai S (eds), *ASIACRYPT 2019*, Springer LNCS, vol. 11922, pp. 273–292, 2019. https://doi.org/10.1007/978-3-030-34621-8_10
16. Huang, Y., Jin, Y., Hu, Z., Zhang, F.: Optimizing the evaluation of l-isogenous curve for isogeny-based cryptography. Information Processing Letters, vol. 178, p. 106301, 2022. 10.1016/j.ipl.2022.106301
17. Gidney, C.: Halving the cost of quantum addition. *Quantum*, vol. 2, p. 74, 2018. https://doi.org/10.22331/q-2018-06-18-74
18. Schroeppel, R., Shamir, A.: A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM Journal on Computing*, vol. 10, no. 3, pp. 456–464, 1981. https://doi.org/10.1137/0210033

19. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: Efficient Isogeny Based Signatures through Class Group Computations. In: *ASIACRYPT 2019*, Springer LNCS, vol. 11921, pp. 227–247, 2019.