Constructing elliptic curve isogenies in quantum subexponential time

Andrew M. Childs[1,2], David Jao[1], and Vladimir Soukharev[1]

[1] Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada
[2] Institute for Quantum Computing, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada

{amchilds,djao,vsoukhar}@math.uwaterloo.ca

Abstract. Given two elliptic curves over a finite field having the same cardinality and endomorphism ring, it is known that the curves admit an isogeny between them, but finding such an isogeny is believed to be computationally difficult. The fastest known classical algorithm takes exponential time, and prior to our work no faster quantum algorithm was known. Recently, public-key cryptosystems based on the presumed hardness of this problem have been proposed as candidates for post-quantum cryptography. In this paper, we give a subexponential-time quantum algorithm for constructing isogenies, assuming the Generalized Riemann Hypothesis (but with no other assumptions). This result suggests that isogeny-based cryptosystems may be uncompetitive with more mainstream quantum-resistant cryptosystems such as lattice-based cryptosystems. As part of our algorithm, we also obtain a second result of independent interest: we provide a new subexponential-time classical algorithm for evaluating a horizontal isogeny given its kernel ideal, assuming (only) GRH, eliminating the heuristic assumptions required by prior algorithms.

1 Introduction

Consider the problem of constructing an isogeny between two given isogenous ordinary elliptic curves defined over a finite field $\mathbb{F}_q$ and having the same endomorphism ring. (For ease of exposition, we say that two such curves are *endomorphic*.) This problem has led to several applications in elliptic curve cryptography, both constructive and destructive. The fastest known probabilistic algorithm for solving this problem is that of Galbraith, Hess, and Smart [12]. Their algorithm is exponential, with a worst-case (and average-case) running time roughly proportional to $\sqrt[4]{q}$.

Although quantum computers are known to compromise several cryptographic protocols of an algebraic nature [9, 13, 30], until now there has been no nontrivial quantum algorithm for constructing isogenies. Consequently, public-key cryptosystems based on isogenies have been proposed from time to time, beginning with a proposal of Couveignes [7]. More recently, Rostovtsev and Stolbunov [25] and Stolbunov [32] proposed refined versions of these cryptosystems with the specific aim of obtaining cryptographic protocols that resist attacks by quantum computers.

In this work, we give a subexponential-time quantum algorithm for constructing an isogeny between two given endomorphic elliptic curves, and show that the running time of our algorithm is bounded above by $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$ under the Generalized Riemann Hypothesis (GRH). This result raises serious questions about the viability of isogeny-based cryptosystems in the context of quantum computers. Indeed, prior implementation results [32, Table 1] indicate that the performance of isogeny-based cryptosystems is already marginal even when only accounting for previously known classical attacks. Hence, even though subexponential attacks in general are not necessarily fatal to a cryptosystem (as demonstrated by the widespread popularity of RSA), the poor performance of isogeny-based cryptosystems suggests that they may be uncompetitive with other approaches

such as lattice-based cryptography [21], for which quantum attacks are not known to offer any advantage.

Our algorithm works by reducing the problem of isogeny construction to the abelian hidden shift problem (see Section 5). By an algorithm of Kuperberg [18], the abelian hidden shift problem can be solved using a subexponential number of queries to certain functions. However, prior to our work there was no known subexponential-time algorithm to evaluate the functions arising in the application to isogenies. Our main technical contribution, described in Section 4, is a subexponential-time (classical) algorithm to compute a certain group action on a set of endomorphic elliptic curves, thereby giving a subexponential-time reduction to the hidden shift problem.

Kuperberg's algorithm for the abelian hidden shift problem uses superpolynomial space, so the same is true of the most straightforward version of our algorithm. However, we also obtain an algorithm using polynomial space by taking advantage of an alternative approach to the abelian hidden shift problem introduced by Regev [22]. Note that Regev only explicitly considered the case of the hidden shift problem in a cyclic group whose order is a power of 2, and even in that case did not compute the constant in the exponent of the running time. As a side result, we fill both of these gaps in the Appendix, showing that the hidden shift problem in any finite abelian group A can be solved in time $L_{|A|}(\frac{1}{2}, \sqrt{2})$ by a quantum computer using only polynomial space. Consequently, we give a polynomial-space quantum algorithm for isogeny construction using time $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2} + \sqrt{2})$. The group relevant to isogeny construction is not always cyclic, so this extension is necessary for our application.

Another contribution of this paper, described in Section 7, is a new (classical) probabilistic algorithm for evaluating the isogeny corresponding to a given kernel ideal and a given starting curve, whose running time is provably subexponential under GRH. In contrast to previous such algorithms, our analysis is conditional solely upon GRH, with no additional assumptions. This result may be of independent interest in the computational theory of elliptic curves and elliptic curve cryptography.

2 Isogenies

Let E and E' be elliptic curves defined over a field F. An *isogeny* $\phi\colon E \to E'$ is an algebraic morphism satisfying $\phi(\infty) = \infty$ (equivalently, $\phi(P+Q) = \phi(P) + \phi(Q)$ for arbitrary points $P, Q \in E$). The *degree* of an isogeny is its degree as an algebraic map. The *endomorphism ring* $\mathrm{End}(E)$ is the set of isogenies from $E(\bar{F})$ to itself, together with the constant homomorphism. This set forms a ring under pointwise addition and composition.

When F is a finite field, the rank of $\mathrm{End}(E)$ as a $\mathbb{Z}$-module is either 2 or 4. We say E is *supersingular* if the rank is 4, and *ordinary* otherwise. A supersingular curve cannot be isogenous to an ordinary curve. Supersingular curves are relatively rare, both in mathematics (they have density zero [28]) and in cryptography (they are less secure, because of the MOV reduction [20]). Thus, we assume in this paper that all curves are ordinary.

An isogeny $\phi\colon E \to E'$ is *separable* or *inseparable* according to whether $K(E)/\phi^*(K(E'))$ is separable or inseparable. The inseparable part of any isogeny is easy to classify and evaluate, since every inseparable isogeny factors through the Frobenius map [31, II.2.12]. Hence we only consider separable isogenies.

Over a finite field $\mathbb{F}_q$, two elliptic curves E and E' are isogenous if and only if $\#E(\mathbb{F}_q) = \#E'(\mathbb{F}_q)$ [33]. We say that two isogenous ordinary elliptic curves over a finite field are *endomorphic*

if their endomorphism rings are equal. The endomorphism ring of an ordinary elliptic curve over a finite field is an imaginary quadratic order $\mathcal{O}_\Delta$ of discriminant $\Delta < 0$. The set of all isomorphism classes (over $\bar{\mathbb{F}}_q$) of endomorphic curves with endomorphism ring $\mathcal{O}_\Delta$ is denoted $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$, where n is the cardinality of any such curve. We represent elements of $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ by taking the j-invariant of any representative curve in the isomorphism class.

An isogeny between endomorphic curves is called a *horizontal* isogeny. Any separable horizontal isogeny $\phi\colon E \to E'$ between curves in $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ can be specified, up to isomorphism, by giving E and $\ker\phi$ [31, III.4.12]. The kernel of an isogeny, in turn, can be represented as an ideal in $\mathcal{O}_\Delta$ [35, Thm. 4.5]. Denote by $\phi_\mathfrak{b}\colon E \to E_\mathfrak{b}$ the isogeny corresponding to an ideal $\mathfrak{b}$ (keeping in mind that $\phi_\mathfrak{b}$ is only defined up to isomorphism of $E_\mathfrak{b}$). Principal ideals correspond to isomorphisms, so any other ideal equivalent to $\mathfrak{b}$ in the ideal class group $\mathrm{Cl}(\mathcal{O}_\Delta)$ of $\mathcal{O}_\Delta$ induces the same isogeny, up to isomorphism [35, Thm. 3.11]. Hence one obtains a well-defined group action

$$\begin{aligned} *\colon \mathrm{Cl}(\mathcal{O}_\Delta) \times \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta) &\to \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta) \\ [\mathfrak{b}] * j(E) &= j(E_\mathfrak{b}) \end{aligned}$$

where $[\mathfrak{b}]$ denotes the ideal class of $\mathfrak{b}$. This group action is free and transitive [35, Thm. 4.5], and thus $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ forms a principal homogeneous space over $\mathrm{Cl}(\mathcal{O}_\Delta)$.

Isogeny graphs under GRH

Our runtime analysis in Section 4 relies on the following result which states, roughly, that random short products of small primes in $\mathrm{Cl}(\mathcal{O}_\Delta)$ yield nearly uniformly random elements of $\mathrm{Cl}(\mathcal{O}_\Delta)$, under GRH.

Theorem 2.1. *Let $\mathcal{O}_\Delta$ be an imaginary quadratic order of discriminant $\Delta < 0$ and conductor c. Set $G = \mathrm{Cl}(\mathcal{O}_\Delta)$. Let B and x be real numbers satisfying $B > 2$ and $x \geq (\ln|\Delta|)^B$. Let S_x be the multiset $A \cup A^{-1}$ where*

$$A = \{[\mathfrak{p}] \in G : \gcd(c, \mathfrak{p}) = 1 \text{ and } N(\mathfrak{p}) \leq x \text{ is prime}\}$$

with $N(\mathfrak{p})$ denoting the norm of $\mathfrak{p}$. Then, assuming GRH, there exists a positive absolute constant $C > 1$, depending only on B, such that for all Δ, a random walk of length

$$t \geq C\frac{\ln|G|}{\ln\ln|\Delta|}$$

in the Cayley graph $\mathrm{Cay}(G, S_x)$ from any starting vertex lands in any fixed subset $S \subset G$ with probability at least $\frac{1}{2}\frac{|S|}{|G|}$.

Proof. Apply Corollary 1.3 of [15] with the parameters

- $K =$ the field of fractions of $\mathcal{O}_\Delta$
- $G = \mathrm{Cl}(\mathcal{O}_\Delta)$
- $q = |\Delta|$.

Observe that by Remark 1.2(a) of [15], Corollary 1.3 of [15] applies to the ring class group $G = \mathrm{Cl}(\mathcal{O}_\Delta)$, since ring class groups are quotients of narrow ray class groups [8, p. 160]. By Corollary 1.3 of [15], Theorem 2.1 holds for all sufficiently large values of $|\Delta|$, i.e., for all but finitely many $|\Delta|$. To prove the theorem for all $|\Delta|$, simply take a larger (but still finite) value of C.

Corollary 2.2 *Theorem 2.1 holds even if the definition of the set A is changed to*

$$A = \{[\mathfrak{p}] \in G : \gcd(m\Delta, \mathfrak{p}) = 1 \text{ and } N(\mathfrak{p}) \leq x \text{ is prime}\}$$

where m is any integer having at most $O(x^{1/2-\varepsilon} \log |\Delta|)$ prime divisors.

Proof. The alternative definition of the set A differs from the original definition by no more than $O(x^{1/2-\varepsilon} \log |\Delta|)$ primes. As indicated in [15, p. 1497], the contribution of these primes can be absorbed into the error term $O(x^{1/2} \log(x) \log(xq))$, and hence does not affect the conclusion of the theorem.

3 Isogeny-based cryptosystems

Given $j(E)$ and $j(E')$, finding an ideal class $[\mathfrak{b}] \in \mathrm{Cl}(\mathcal{O}_\Delta)$ such that $[\mathfrak{b}] * j(E) = j(E')$ (i.e., the so-called *quotient* of $j(E)$ and $j(E')$) appears to be a hard problem, with the fastest known algorithm to date using $O(q^{1/4})$ operations [12]. This fact has led to the design of public-key cryptosystems and protocols based on the conjectured infeasibility of finding quotients in $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$.

Isogeny-based cryptosystems first appeared in an unpublished manuscript of Couveignes [7], where the author proposes homogeneous spaces in general as an alternative to (and generalization of) discrete logarithms over groups, and cites as a specific example the space $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ [7, §5.1]. More recently, the idea of cryptosystems based on homogeneous spaces and isogenies between endomorphic curves has been revived by Rostovtsev and Stolbunov [25] and Stolbunov [32], with resistance to quantum attacks being cited as an explicit design goal [25, §11][32, §7.2]. We are not concerned here with the details of their cryptographic protocols; for our purposes, it suffices to note that the ability to evaluate quotients in $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ leads to a total break of the protocols (i.e., recovery of the private key).

An interesting property of isogeny-based cryptosystems is that they do not require the ability to evaluate the group action $*$ efficiently on arbitrary inputs. It is enough to sample from random smooth ideals (for which $*$ can be evaluated efficiently) when performing operations such as key generation [32, §6.2]. However, in order to attack these cryptosystems using our approach, we *do* require the ability to evaluate arbitrary instances of $[\mathfrak{b}] * j(E)$. We turn to this problem in the next section.

4 Computing the action of $\mathrm{Cl}(\mathcal{O}_\Delta)$ on $\mathrm{Ell}(\mathcal{O}_\Delta)$

In this section, we describe a new classical (i.e., non-quantum) algorithm to evaluate the group action $[\mathfrak{b}] * j(E)$ of $\mathrm{Cl}(\mathcal{O}_\Delta)$ on $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$, where $\Delta < 0$ is an imaginary quadratic discriminant. The algorithm takes as input an ideal class $[\mathfrak{b}]$ of $\mathcal{O}_\Delta$ and an isomorphism class of elliptic curves (represented by their j-invariant), and outputs a j-invariant $j(E') \in \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$.

Throughout this section, E is an ordinary elliptic curve defined over a finite field $\mathbb{F}_q$ of cardinality n and endomorphism ring $\mathcal{O}_\Delta$. We are given an ideal class $[\mathfrak{b}]$ in $\mathrm{Cl}(\mathcal{O}_\Delta)$, and we wish to evaluate $[\mathfrak{b}] * j(E)$. We use the notation

$$L_N(\tfrac{1}{2}, c) = \exp[(c + o(1))\sqrt{\ln N \ln \ln N}].$$

For convenience, we denote $L_{\max\{|\Delta|,q\}}(\frac{1}{2}, c)$ by $L(\frac{1}{2}, c)$.

Algorithm 1 Computing a factor base

Input: Δ, q, and a parameter z
Output: A factor base $\mathcal{F}$, or `nil`

Set $L \leftarrow \lceil L(\frac{1}{2}, z) \rceil$, $k \leftarrow \lceil \ln L \rceil$, $\mathcal{F} \leftarrow \emptyset$
for all primes $p < L$ **do**
  if `kronecker`$(\Delta, p) = 1$ **then**
    $i \leftarrow 0$
    repeat
      $i \leftarrow i + 1$
      $g \leftarrow$ `primeForm`(Δ, p)
    until $i > 2k$ or $g \neq$ `nil`
    if $g \neq$ `nil` **then**
      $\mathcal{F} \leftarrow \mathcal{F} \cup \{g, g^\sigma\}$
    else
      Return `nil`
    end if
  end if
end for
Return $\mathcal{F}$

In Section 4.1 we show that, under GRH, our algorithm has a running time of $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$, which is subexponential in the input size. We stress that although similar algorithms have appeared in several previous works, our algorithm is the first to achieve provably subexponential running time without appealing to any conditional hypotheses other than GRH. Section 8 discusses the relationship between our work and earlier work on this problem.

We present our algorithm in several stages.

Computing a factor base. Algorithm 1 computes a factor base for $\mathrm{Cl}(\mathcal{O}_\Delta)$ consisting of all split primes up to $L(\frac{1}{2}, z)$. The optimal value of the parameter z is determined in Section 4.1. The algorithm is based on, and indeed almost identical to, Algorithm 11.1 in [5]. The subroutine `primeForm` [5, §3.4] calculates a quadratic form corresponding to a prime ideal of norm p, and the subroutine `kronecker` [5, §3.4.3] calculates the Kronecker symbol. The map σ denotes complex conjugation.

Computing a relation. Given an ideal class $[\mathfrak{b}] \in \mathrm{Cl}(\mathcal{O}_\Delta)$, Algorithm 2 produces a relation vector $\mathbf{z} = (z_1, \ldots, z_f) \in \mathbb{Z}^f$ for $[\mathfrak{b}]$, with respect to a factor base $\mathcal{F} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_f\}$, satisfying $[\mathfrak{b}] = \mathcal{F}^{\mathbf{z}} := \mathfrak{p}_1^{z_1} \cdots \mathfrak{p}_f^{z_f}$, with the additional property that the L^∞-norm $|\mathbf{z}|_\infty$ of $\mathbf{z}$ is less than $O(\ln |\Delta|)$ for some absolute implied constant (cf. Proposition 4.6). It is similar to Algorithm 11.2 in [5], except that we impose a constraint on $|\mathbf{v}|_\infty$ in Step 2 in order to keep $|\mathbf{z}|_\infty$ small, and (for performance reasons) we use Bernstein's algorithm instead of trial division to find smooth elements.

We remark that Corollary 9.3.12 of [5] together with the restriction $C > 1$ in Theorem 2.1 implies that there exists a value of t satisfying the inequality in Algorithm 2.

Computing $j(E')$. Algorithm 3 is the main algorithm of this section. It takes as input a discriminant $\Delta < 0$, an ideal class $[\mathfrak{b}] \in \mathrm{Cl}(\mathcal{O}_\Delta)$, and a j-invariant $j(E) \in \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ where q is relatively prime to 6, and produces the element $j(E') \in \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ such that $[\mathfrak{b}] * j(E) = j(E')$. Eliminating the primes dividing qn is necessary for the computation of the isogenies in the final step of the algorithm.

Algorithm 3 is correct since the ideals $\mathfrak{b}$ and $\mathcal{F}^{\mathbf{z}}$ belong to the same ideal class, and thus act identically on $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$.

Algorithm 2 Computing a relation

Input: Δ, q, n, z, $[\mathfrak{b}]$, and an integer t satisfying $C\frac{\ln|\mathrm{Cl}(\mathcal{O}_\Delta)|}{\ln\ln|\Delta|} \leq t \leq C\ln|\Delta|$ where C is the constant of Theorem 2.1/Corollary 2.2

Output: A relation vector $\mathbf{z} \in \mathbb{Z}^f$ such that $[\mathfrak{b}] = [\mathcal{F}^{\mathbf{z}}]$, or `nil`

1: Using Algorithm 1, compute a factor base. Discard any primes dividing qn to obtain a new factor base $\mathcal{F} = \{\mathfrak{p}_1, \mathfrak{p}_2, \ldots, \mathfrak{p}_f\}$
2: Set $\mathcal{S} \leftarrow \emptyset$, $\mathcal{P} \leftarrow \{N(\mathfrak{p}) : \mathfrak{p} \in \mathcal{F}\}$
3: Set $\ell \leftarrow L(\frac{1}{2}, \frac{1}{4z})$
4: **for** $i = 0$ to ℓ **do**
5: Select $\mathbf{v} \in \mathbb{Z}^f_{0..|\Delta|-1}$ uniformly at random subject to the condition that $|\mathbf{v}|_\infty = t$
6: Calculate the reduced ideal $\mathfrak{a}_{\mathbf{v}}$ in the ideal class $[\mathfrak{b}] \cdot [\mathcal{F}^{\mathbf{v}}]$
7: Set $\mathcal{S} \leftarrow \mathcal{S} \cup N(\mathfrak{a}_{\mathbf{v}})$
8: **end for**
9: Using Bernstein's algorithm [2], find a $\mathcal{P}$-smooth element $N(\mathfrak{a}_{\mathbf{v}}) \in \mathcal{S}$ (if there exists one), or else return `nil`
10: Find the prime factorization of the integer $N(\mathfrak{a}_{\mathbf{v}})$
11: Using Seysen's algorithm [29, Thm. 3.1] on the prime factorization of $N(\mathfrak{a}_{\mathbf{v}})$, factor the ideal $\mathfrak{a}_{\mathbf{v}}$ over $\mathcal{F}$ to obtain $\mathfrak{a}_{\mathbf{v}} = \mathcal{F}^{\mathbf{a}}$ for some $\mathbf{a} \in \mathbb{Z}^f$
12: Return $\mathbf{z} = \mathbf{a} - \mathbf{v}$

Algorithm 3 Computing $j(E')$

Input: Δ, q, $[\mathfrak{b}]$, and a j-invariant $j(E) \in \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$

Output: The element $j(E') \in \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ such that $[\mathfrak{b}] * j(E) = j(E')$

1: Using Algorithm 2 with any valid choice of t, compute a relation $\mathbf{z} \in \mathbb{Z}^f$ such that $[\mathfrak{b}] = [\mathcal{F}^{\mathbf{z}}] = [\mathfrak{p}_1^{z_1}\mathfrak{p}_2^{z_2}\cdots\mathfrak{p}_f^{z_f}]$
2: Compute a sequence of isogenies $(\phi_1, \ldots, \phi_s)$ such that the composition $\phi_c \colon E \to E_c$ of the sequence has kernel $E[\mathfrak{p}_1^{z_1}\mathfrak{p}_2^{z_2}\cdots\mathfrak{p}_f^{z_f}]$, using the method of [4, §3]
3: Return $j(E_c)$

4.1 Runtime analysis

Here we determine the theoretical running time of Algorithm 3, as well as the optimal value of the parameter z in Algorithm 1. As is typical for subexponential-time factorization algorithms involving a factor base, these two quantities depend on each other, and hence both are calculated simultaneously.

Proposition 4.1 *Algorithm 1 takes time $L(\frac{1}{2}, z)$ and succeeds with probability at least $1/4$.*

Proof. Since the body of Algorithm 1 is identical to Algorithm 11.1 in [5], the proposition follows from Lemmas 11.3.1 and 11.3.2 of [5].

Proposition 4.2 *The running time of Algorithm 2 is at most $L(\frac{1}{2}, z) + L(\frac{1}{2}, \frac{1}{4z})$, assuming GRH.*

Proof. Step 2 of the algorithm requires $L(\frac{1}{2}, z)$ norm computations. Step 3 is negligible. Step 6 requires $C\ln|\Delta|$ multiplications in the class group, each of which requires $O((\ln|\Delta|)^{1+\varepsilon})$ bit operations [26]. Hence the **for** loop in Steps 4–8 has running time $L(\frac{1}{2}, \frac{1}{4z}) \cdot O((\ln|\Delta|)^{2+\varepsilon})$. Bernstein's algorithm [2] in Step 9 has a running time of $b(\log_2 b)^{2+\varepsilon}$ where $b = L(\frac{1}{2}, z) + L(\frac{1}{2}, \frac{1}{4z})$ is the combined size of $\mathcal{S}$ and $\mathcal{P}$. Finding the prime factorization in Step 10 costs $L(\frac{1}{2}, z)$ using trial division, and Seysen's algorithm [29, Thm. 3.1] in Step 11 has negligible cost under ERH (and hence GRH). Accordingly, we find that the running time is

$$L(\tfrac{1}{2}, z) + O((\ln|\Delta|)^{2+\varepsilon}) \cdot L(\tfrac{1}{2}, \tfrac{1}{4z}) + b(\log_2 b)^{2+\varepsilon} + L(\tfrac{1}{2}, z) = L(\tfrac{1}{2}, z) + L(\tfrac{1}{2}, \tfrac{1}{4z}),$$

as desired.

Remark 4.3. If we use quantum algorithms, then the performance boost obtained from Bernstein's algorithm is not necessary, since quantum computers can factor integers in polynomial time [30]. This allows for some simplification in Algorithm 2 in the quantum setting: there is no need to store elements of $\mathcal{S}$ (since one can test directly for smooth integers via factoring), and the algorithm no longer requires superpolynomial space.

Proposition 4.4 *Under GRH, the probability that a single iteration of the* ***for*** *loop of Algorithm 2 produces an $\mathcal{F}$-smooth ideal $\mathfrak{a}_{\mathbf{v}}$ is at least $L(\frac{1}{2}, -\frac{1}{4z})$.*

Proof. We adopt the notation used in Theorem 2.1 and Corollary 2.2. Apply Corollary 2.2 with the values $m = qn$, $B = 3$, and $x = f = L(\frac{1}{2}, z) \gg (\ln|\Delta|)^B$. Observe that m has at most $O(\log q)$ prime divisors, and

$$O(\log q) \ll L_q(\tfrac{1}{2}, z(\tfrac{1}{2} - \varepsilon)) \leq L(\tfrac{1}{2}, z(\tfrac{1}{2} - \varepsilon)) = x^{1/2-\varepsilon}.$$

Therefore Corollary 2.2 applies. The ideal class $[\mathfrak{b}] \cdot [\mathcal{F}^{\mathbf{v}}]$ is equal to the ideal class obtained by taking the walk of length t in the Cayley graph $\mathrm{Cay}(G, S_x)$, having initial vertex $[\mathfrak{b}]$, and whose edges correspond to the nonzero coordinates of the vector $\mathbf{v}$. Hence a random choice of vector $\mathbf{v}$ under the constraints of Algorithm 2 yields the same probability distribution as a random walk in $\mathrm{Cay}(G, S_x)$ starting from $[\mathfrak{b}]$.

Let S be the set of reduced ideals in G with $L(\frac{1}{2}, z)$-smooth norm. By [5, Lemma 11.4.4], $|S| \geq \sqrt{|\Delta|} L_{|\Delta|}(\frac{1}{2}, -\frac{1}{4z}) \geq \sqrt{|\Delta|} L(\frac{1}{2}, -\frac{1}{4z})$. Hence, by Corollary 2.2, the probability that $\mathfrak{a}_{\mathbf{v}}$ lies in S is at least

$$\frac{1}{2}\frac{|S|}{|G|} \geq \frac{1}{2} \cdot \frac{\sqrt{|\Delta|}}{|G|} \cdot L(\tfrac{1}{2}, -\tfrac{1}{4z}).$$

Finally, Theorem 9.3.11 of [5] states that $\frac{\sqrt{|\Delta|}}{|G|} \geq \frac{1}{\ln|\Delta|}$. Hence the probability that $\mathfrak{a}_{\mathbf{v}}$ is $\mathcal{F}$-smooth is at least

$$\frac{1}{2} \cdot \frac{1}{\ln|\Delta|} \cdot L(\tfrac{1}{2}, -\tfrac{1}{4z}) = L(\tfrac{1}{2}, -\tfrac{1}{4z}).$$

The result follows.

Corollary 4.5 *Under GRH, the probability that Algorithm 2 succeeds is at least $1 - \frac{1}{e}$.*

Proof. Algorithm 2 loops through $\ell = L(\frac{1}{2}, \frac{1}{4z})$ vectors $\mathbf{v}$, and by Proposition 4.4, each such choice of $\mathbf{v}$ has an independent $1/\ell$ chance of producing a smooth ideal $\mathfrak{a}_{\mathbf{v}}$. Therefore the probability of success is at least

$$1 - \left(1 - \frac{1}{\ell}\right)^{\ell} > 1 - \frac{1}{e},$$

as desired.

The following proposition shows that the relation vector $\mathbf{z}$ produced by Algorithm 2 is guaranteed to have small coefficients.

Proposition 4.6 *Any vector $\mathbf{z}$ output by Algorithm 2 satisfies $|\mathbf{z}|_\infty < (C+1)\ln|\Delta|$.*

Proof. Since $\mathbf{z} = \mathbf{a} - \mathbf{v}$, we have $|\mathbf{z}|_\infty \leq |\mathbf{a}|_\infty + |\mathbf{v}|_\infty$. But $|\mathbf{v}|_\infty \leq C \ln |\Delta|$ by construction, and the norm of $\mathfrak{a}_v$ is less than $\sqrt{|\Delta|/3}$ [5, Prop. 9.1.7], which implies

$$|\mathbf{a}|_\infty < \log_2 \sqrt{|\Delta|/3} < \log_2 \sqrt{|\Delta|} < \ln |\Delta|.$$

This completes the proof.

Finally, we analyze the running time of Algorithm 3.

Theorem 4.7. *Under GRH, Algorithm 3 succeeds with probability at least* $\frac{1}{4}(1 - \frac{1}{e})$ *and runs in time at most*

$$L(\tfrac{1}{2}, \tfrac{1}{4z}) + \max\{L(\tfrac{1}{2}, 3z), L(\tfrac{1}{2}, z)(\ln q)^{3+\varepsilon}\}.$$

Proof. We have shown that Algorithm 1 has running time $L(\frac{1}{2}, z)$ and success probability at least $1/4$, and Algorithm 2 has running time $L(\frac{1}{2}, z) + L(\frac{1}{2}, \frac{1}{4z})$ and success probability at least $1 - \frac{1}{e}$. Assuming that both these algorithms succeed, the computation of the individual isogenies ϕ_i in Step 2 of Algorithm 3 proceeds in one of two ways, depending on whether the characteristic of $\mathbb{F}_q$ is large [4, §3.1] or small [4, §3.2]. The large characteristic algorithm fails when the characteristic is small, whereas the small characteristic algorithm succeeds in all situations, but is slightly slower in large characteristic. For simplicity, we consider only the more general algorithm.

The general algorithm proceeds in two steps. In the first step, we compute the kernel polynomial of the isogeny. The time to perform one such calculation is $O((\ell(\ln q)\max(\ell, \ln q)^2)^{1+\varepsilon})$ in all cases ([19, Thm. 1] for characteristic ≥ 5 and [10, Thm. 1] for characteristic 2 or 3). In the second step, we evaluate the isogeny using Vélu's formulae [34]. This second step has a running time of $O(\ell^{2+\varepsilon}(\ln q)^{1+\varepsilon})$ [14, p. 214]. Hence the running time of Step 2 is at most

$$|\mathbf{z}|_\infty (O((\ell(\ln q)\max(\ell, \ln q)^2)^{1+\varepsilon}) + O(\ell^{2+\varepsilon}(\ln q)^{1+\varepsilon})).$$

By Proposition 4.6, this expression is at most

$$\begin{aligned}&(C+1)(\ln |\Delta|)(\max\{L(\tfrac{1}{2}, 3z), L(\tfrac{1}{2}, z)(\ln q)^{3+\varepsilon}\} + L(\tfrac{1}{2}, 2z)(\ln q)^{1+\varepsilon}) \\ &= \max\{L(\tfrac{1}{2}, 3z), L(\tfrac{1}{2}, z)(\ln q)^{3+\varepsilon}\}.\end{aligned}$$

The theorem follows.

Corollary 4.8 *Under GRH, Algorithm 3 has a worst-case running time of at most* $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$.

Proof. Using the inequality $|\Delta| \leq 4q$, we may rewrite Theorem 4.7 in terms of q. We obtain

$$L(\tfrac{1}{2}, \tfrac{1}{4z}) + \max\{L(\tfrac{1}{2}, 3z), L(\tfrac{1}{2}, z)(\ln q)^{3+\varepsilon}\} \leq L_q(\tfrac{1}{2}, \tfrac{1}{4z} + 3z).$$

The optimal choice of $z = \frac{1}{2\sqrt{3}}$ yields the running time bound of $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$.

5 A quantum algorithm for constructing isogenies

Our quantum algorithm for constructing isogenies uses a reduction to the abelian hidden shift problem. To define this problem, let A be a known finite abelian group (with the group operation written multiplicatively) and let $f_0, f_1 : A \to S$ be black-box functions, where S is a known finite set. We say that f_0, f_1 hide a shift $s \in A$ if f_0 is injective and $f_1(x) = f_0(xs)$ (i.e., f_1 is a shifted version of f_0). The goal of the hidden shift problem is to determine s using queries to such black-box functions. Note that this problem is equivalent to the hidden subgroup problem in the A-dihedral group, the nonabelian group $A \rtimes \mathbb{Z}_2$ where $\mathbb{Z}_2$ acts on A by inversion.

Isogeny construction is easily reduced to the hidden shift problem using the group action defined in Section 2. Given endomorphic curves E_0, E_1 with endomorphism ring $\mathcal{O}_\Delta$, we define functions $f_0, f_1 : \mathrm{Cl}(\mathcal{O}_\Delta) \to \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ that hide $[\mathfrak{s}] \in \mathrm{Cl}(\mathcal{O}_\Delta)$, where $[\mathfrak{s}]$ is the ideal class such that $[\mathfrak{s}] * j(E_0) = j(E_1)$. Specifically, let $f_c([\mathfrak{b}]) = [\mathfrak{b}] * j(E_c)$. Then it is immediate that f_0, f_1 hide $[\mathfrak{s}]$.

Lemma 5.1. *The function f_0 is injective and $f_1([\mathfrak{b}]) = f_0([\mathfrak{b}][\mathfrak{s}])$.*

Proof. Since $*$ is a group action,

$$\begin{aligned} f_1([\mathfrak{b}]) &= [\mathfrak{b}] * j(E_1) \\ &= [\mathfrak{b}] * ([\mathfrak{s}] * j(E_0)) \\ &= ([\mathfrak{b}][\mathfrak{s}]) * j(E_0) \\ &= f_0([\mathfrak{b}][\mathfrak{s}]). \end{aligned}$$

If there are distinct ideal classes $[\mathfrak{b}], [\mathfrak{b}']$ such that $f_0([\mathfrak{b}]) = f_0([\mathfrak{b}'])$, then $[\mathfrak{b}] * j(E_0) = [\mathfrak{b}'] * j(E_0)$, which contradicts the fact that the action is free and transitive [35, Thm. 4.5]. Thus f_0 is injective.

This reduction allows us to apply quantum algorithms for the hidden shift problem to construct isogenies. The hidden shift problem can be solved in quantum subexponential time assuming we can evaluate the group action in subexponential time. The latter is possible due to Algorithm 3.

We consider two different approaches to solving the hidden shift problem in subexponential time on a quantum computer. The first, due to Kuperberg [18], has a faster running time but requires superpolynomial space. The second approach generalizes an algorithm of Regev [22]. It uses only polynomial space, but is slower than Kuperberg's original algorithm.

Method 1: Kuperberg's algorithm. Kuperberg's approach to the abelian hidden shift problem is based on the idea of performing a Clebsch-Gordan sieve on coset states. The following appears as Theorem 7.1 of [18].

Theorem 5.1. *The abelian hidden shift problem has a [quantum] algorithm with time and query complexity $2^{O(\sqrt{n})}$, where n is the length of the output, uniformly for all finitely generated abelian groups.*

In our context, $2^{O(\sqrt{n})} = 2^{O(\sqrt{\ln|\Delta|})}$ since $|\mathrm{Cl}(\mathcal{O}_\Delta)| = O(\sqrt{\Delta}\ln\Delta)$ [5, Thm. 9.3.11]. Furthermore, $2^{O(\sqrt{\ln|\Delta|})} = L(\frac{1}{2}, o(1)) = L(\frac{1}{2}, 0)$ regardless of the value of the implied constant in the exponent, since the exponent on the left has no $\sqrt{\ln\ln|\Delta|}$ term, whereas $L(\frac{1}{2}, 0)$ does. As mentioned above, Kuperberg's algorithm also requires superpolynomial space (specifically, $2^{O(\sqrt{n})}$ space).

Algorithm 4 Isogeny construction

Input: A finite field $\mathbb{F}_q$, a discriminant $\Delta < 0$, and Weierstrass equations of endomorphic elliptic curves E_0, E_1
Output: $[\mathfrak{s}] \in \mathrm{Cl}(\mathcal{O}_\Delta)$ such that $[\mathfrak{s}] * j(E_0) = j(E_1)$
1: Decompose $\mathrm{Cl}(\mathcal{O}_\Delta) = \langle [\mathfrak{b}_1] \rangle \oplus \cdots \oplus \langle [\mathfrak{b}_k] \rangle$ where $|\langle [\mathfrak{b}_j] \rangle| = n_j$
2: Solve the hidden shift problem defined by functions $f_0, f_1 : \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k} \to \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ satisfying $f_c(x_1, \ldots, x_k) = ([\mathfrak{b}_1]^{x_1} \cdots [\mathfrak{b}_k]^{x_k}) * j(E_c)$, giving some $(s_1, \ldots, s_k) \in \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$
3: Output $[\mathfrak{s}] = [\mathfrak{b}_1]^{s_1} \cdots [\mathfrak{b}_k]^{s_k}$

Method 2: Regev's algorithm. Regev [22] showed that a variant of Kuperberg's sieve leads to a slightly slower algorithm using only polynomial space. In particular, he proved Theorem 5.2 below in the case where A is a cyclic group whose order is a power of 2 (without giving an explicit value for the constant in the exponent). Theorem 5.2 generalizes Regev's algorithm to arbitrary finite abelian groups. A detailed proof of Theorem 5.2 appears in the Appendix (see Theorem A.1).

Theorem 5.2. *Let A be a finite abelian group and let functions f_0, f_1 hide some unknown $s \in A$. Then there is a quantum algorithm that finds s with time and query complexity $L_{|A|}(\frac{1}{2}, \sqrt{2})$ using space* $\mathrm{poly}(\log |A|)$.

We now return to the original problem of constructing isogenies. Note that to use the hidden shift approach, the group $\mathrm{Cl}(\mathcal{O}_\Delta)$ must be known. Furthermore, we need to invoke Algorithm 3, which assumes that Δ is known. Thus, we assume in this section that the discriminant Δ is given as part of the input. This requirement poses no difficulty, since all the cryptosystems described in Section 3 (namely, those of [7, 25, 32]) stipulate that $\mathcal{O}_\Delta$ is a maximal order, in which case its discriminant can be computed easily: simply calculate the number of points $\#E$ on the curve (and hence the trace $t(E) := q+1-\#E$) using Schoof's algorithm [27], factor $t(E)^2 - 4q$, and divide this value by its largest square factor to obtain Δ. For the sake of completeness, we show in Section 6 how to handle the case of unknown Δ without increasing the asymptotic running time.

Assuming Δ is known, we decompose $\mathrm{Cl}(\mathcal{O}_\Delta)$ as a direct sum of cyclic groups, with a known generator for each, and then solve the hidden shift problem. The overall procedure is described in Algorithm 4.

Theorem 5.3. *Assuming GRH, Algorithm 4 runs in time $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$ (respectively, $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2} + \sqrt{2})$) using Theorem 5.1 (respectively, Theorem 5.2) to solve the hidden shift problem.*

Proof. We perform Step 1 using [6, Algorithm 10], which determines the structure of an abelian group given a generating set and a unique representation for the group elements. We represent the elements uniquely using reduced quadratic forms, and we use the fact that, under ERH (and hence GRH), the set of ideal classes of norm at most $6 \ln^2 |\Delta|$ form a generating set [1, p. 376]. By Theorem 5.1 (resp. Theorem 5.2), Step 2 uses $L(\frac{1}{2}, o(1)) = L(\frac{1}{2}, 0)$ (resp. $L(\frac{1}{2}, \sqrt{2})$) evaluations of the functions f_i. By Corollary 4.8, these functions can be evaluated in time $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$ using Algorithm 3, assuming GRH. Overall, Step 2 takes time $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2} + o(1)) = L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$ if Theorem 5.1 is used, or $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2} + \sqrt{2})$ if Theorem 5.2 is used. The cost of Step 3 is negligible.

Remark 5.4. Note that the running time of the algorithm is ultimately limited by two factors: the best known quantum algorithm for the hidden shift problem takes superpolynomial time, and the best known classical or quantum algorithm for computing the action $*$ is the one described in

Algorithm 3. Improving only one of these results to take polynomial time would still result in a superpolynomial-time algorithm.

Remark 5.5. In general, any cryptosystem whose security relies on the difficulty of finding quotients in a principal homogeneous space can be broken in subexponential time by a quantum computer using our approach, provided that the group action can be evaluted in subexponential time. However, in the specific case of isogeny-based cryptosystems, there was no known method to evaluate the group action in subexponential time on arbitrary inputs prior to this work. Algorithm 3 of Section 4 fills that gap, allowing isogeny computation to be approached as a hidden shift problem.

6 Constructing isogenies when the endomorphism ring is unknown

Algorithm 4 assumes that for the given elliptic curves E_0, E_1, we know the discriminant Δ such that $\mathrm{End}(E_0) = \mathrm{End}(E_1) = \mathcal{O}_\Delta$. If Δ is unknown, we can compute it in time $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$ using [3, Algorithm 1]. However, that algorithm requires extra heuristic assumptions in addition to GRH. In this section we describe how Algorithm 4 can be modified to work when Δ is unknown, assuming only GRH.

Without knowing Δ, we cannot directly perform computations in $\mathrm{Cl}(\mathcal{O}_\Delta)$. However, using Schoof's algorithm [27] to count the points on E_0, we can determine $\hat{\Delta} := t(E_0)^2 - 4q$. Given $\hat{\Delta}$, we can perform computations in $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$. In particular, we define a group action

$$\begin{aligned} \hat{*} : \mathrm{Cl}(\mathcal{O}_{\hat{\Delta}}) \times \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta) &\to \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta) \\ [\mathfrak{b}] \mathbin{\hat{*}} j(E) &= j(E_{\mathfrak{b}}). \end{aligned}$$

This action naturally extends the action $*$ to the larger group $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$. Indeed, consider the function $\pi : \mathrm{Cl}(\mathcal{O}_{\hat{\Delta}}) \to \mathrm{Cl}(\mathcal{O}_\Delta)$ defined as follows: for any $[\mathfrak{b}] \in \mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$, let $\pi([\mathfrak{b}])$ be the ideal class of $\mathfrak{b}\mathcal{O}_\Delta$ in $\mathrm{Cl}(\mathcal{O}_\Delta)$, where $\mathfrak{b}$ is any representative of $[\mathfrak{b}]$ in $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$. The homomorphism π is well-defined and surjective (see [8, Eqn. 7.25] for the case where $\mathcal{O}_\Delta$ is maximal; the general case is similar), and the action $\hat{*}$ factors through $*$, i.e., $[\mathfrak{b}] \mathbin{\hat{*}} j = \pi([\mathfrak{b}]) * j$.

Given $\hat{\Delta}$, we can compute the action $\hat{*}$ using Algorithm 3. Indeed, we simply provide $\hat{\Delta}$ instead of Δ as input to Algorithm 3. Steps 1 and 2 of Algorithm 3 are valid for any discriminant, and hence using $\hat{\Delta}$ presents no difficulties in these steps. In Step 3, we compute the isogenies ϕ_i corresponding to the ideals $\mathfrak{p}_i$ using the method of [4, §3]. We show that this procedure applied to $\mathfrak{p}_i \in \mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$ yields results identical to those obtained from using $\pi(\mathfrak{p}_i) \in \mathrm{Cl}(\mathcal{O}_\Delta)$. As in [4, §3], we write $\mathfrak{p}_i = (\ell, c + d \cdot \mathrm{Frob}_q)$ and $\pi(\mathfrak{p}_i) = (\ell', c' + d' \cdot \mathrm{Frob}_q)$, where Frob_q is the Frobenius map and ℓ (resp. ℓ') is the norm of $\mathfrak{p}_i$ (resp. $\pi(\mathfrak{p}_i)$). Observe that the two ideals $\mathfrak{p}_i$ and $\pi(\mathfrak{p}_i)$ have the same norm [8, Prop. 7.20], and thus $\ell = \ell'$. The values of c and d need not be identical to c' and d', but fortunately these values are used only in one place, namely, to disambiguate between $\mathfrak{p}_i$ and $\bar{\mathfrak{p}}_i$ by computing whether $\mathrm{Frob}_q(P) = (-c/d)P$ for points $P \in \ker \phi_i \subset E[\ell]$. In fact, we claim that the value of $-c/d \bmod \ell$ is preserved by π. To see this, note that the index of $\mathfrak{p}_i$ in $\pi(\mathfrak{p}_i)$ is equal to $f = (\hat{\Delta}/\Delta)^{1/2}$ [8, 7.2 and 7.20]. Hence, multiplying a non-integer generator of $\pi(\mathfrak{p}_i)$ by f yields a non-integer generator of $\mathfrak{p}_i$. Thus we may choose generators such that $c = fc'$ and $d = fd'$. Moreover, f is nonzero mod ℓ (by Step 3 of Algorithm 1). It follows that $-c/d = -fc'/fd' = -c'/d' \bmod \ell$.

Using $\hat{\Delta}$ in place of Δ degrades the performance of the algorithm, since $|\hat{\Delta}| \geq |\Delta|$. However, $|\hat{\Delta}|$ is in any case still bounded by $O(q)$, so the running time bound of $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$ for Algorithm 3 still holds.

Algorithm 5 Isogeny construction with unknown Δ

Input: A finite field $\mathbb{F}_q$ and Weierstrass equations of endomorphic elliptic curves E_0, E_1
Output: $[\hat{\mathfrak{s}}] \in \mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$ such that $[\hat{\mathfrak{s}}] \mathbin{\hat{*}} j(E_0) = j(E_1)$

1: Compute $\hat{\Delta} = t(E_0)^2 - 4q$
2: Decompose $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}}) = \langle [\hat{\mathfrak{b}}_1] \rangle \oplus \cdots \oplus \langle [\hat{\mathfrak{b}}_{\hat{k}}] \rangle$ where $|\langle [\hat{\mathfrak{b}}_j] \rangle| = \hat{n}_j$
3: Find a generating set for $\ker(\pi)$, the stabilizer of $j(E_0)$ with respect to the action $\hat{*}$
4: Decompose $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})/\ker(\pi) = \langle [\mathfrak{b}_1] \ker(\pi) \rangle \oplus \cdots \oplus \langle [\mathfrak{b}_k] \ker(\pi) \rangle$ where $|\langle [\mathfrak{b}_j] \ker(\pi) \rangle| = |\langle \pi([\mathfrak{b}_j]) \rangle| = n_j$
5: Solve the hidden shift problem defined by functions $\hat{f}_0, \hat{f}_1 : \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k} \to \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ satisfying $\hat{f}_c(x_1, \ldots, x_k) = ([\mathfrak{b}_1]^{x_1} \cdots [\mathfrak{b}_k]^{x_k}) \mathbin{\hat{*}} j(E_c)$, giving some $(s_1, \ldots, s_k) \in \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_k}$
6: Output $[\hat{\mathfrak{s}}] = [\mathfrak{b}_1]^{s_1} \cdots [\mathfrak{b}_k]^{s_k}$

We now show how to use $\hat{*}$ to determine $\ker(\pi)$. If we can determine $\ker(\pi)$, then we can effectively perform computations in $\mathrm{Cl}(\mathcal{O}_\Delta) \cong \mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})/\ker(\pi)$. Generators for $\ker(\pi)$ can be found by solving the abelian stabilizer problem defined by the action $\hat{*}$. In the abelian stabilizer problem, we are given an action of an abelian group on a finite set; the goal is to find the stabilizer subgroup of a given set element. Kitaev [17] showed that this problem can be solved by a quantum computer in polynomial time. Since $\ker(\pi)$ is the stabilizer of $j(E)$ for any elliptic curve E with $\mathrm{End}(E) = \mathcal{O}_\Delta$, we can find a generating set for $\ker(\pi)$ in polynomial time. (Note that the abelian stabilizer problem is a special case of the well-known abelian hidden subgroup problem; in particular, the subgroup $\ker(\pi)$ is hidden by the function $f : \mathrm{Cl}(\mathcal{O}_{\hat{\Delta}}) \to \mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ defined by $f([\mathfrak{b}]) = [\mathfrak{b}] \mathbin{\hat{*}} j(E) = j(E_{\mathfrak{b}})$, meaning that f is constant on cosets of $\ker(\pi)$ in $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$ and distinct on different cosets.)

Putting these ideas together, Algorithm 5 shows how to construct isogenies in subexponential time even when Δ is unknown.

Theorem 6.1. *Algorithm 5 runs in time $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$ (resp. $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2} + \sqrt{2})$) using Theorem 5.1 (resp. Theorem 5.2).*

Proof. Step 1 takes polynomial time using Schoof's algorithm [27]. Step 2 takes polynomial time using [6]. Step 3 takes polynomially many queries to f (and polynomially many other operations) using [17]; each query to f can be implemented in time $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$ using the modification of Algorithm 3 described above. In Step 4, although we have generators for $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$ and $\ker(\pi)$, we cannot directly apply [6] since we do not have unique representatives for the elements of the quotient $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})/\ker(\pi)$. However, using the technique of representing cosets by uniform superpositons [36, Section 4.2], there is a polynomial-time quantum algorithm that produces $[\mathfrak{b}_1], \ldots, [\mathfrak{b}_k] \in \mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$ such that $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})/\ker(\pi) = \langle [\mathfrak{b}_1] \ker(\pi) \rangle \oplus \cdots \oplus \langle [\mathfrak{b}_k] \ker(\pi) \rangle$, where $[\mathfrak{b}] \ker(\pi)$ denotes the coset of $\ker \pi$ in $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$ with representative $[\mathfrak{b}]$. This effectively gives a decomposition of the isomorphic group $\mathrm{Cl}(\mathcal{O}_\Delta) = \langle \pi([\mathfrak{b}_1]) \rangle \oplus \cdots \oplus \langle \pi([\mathfrak{b}_k]) \rangle$, except that we cannot efficiently compute π. However, in Step 5 we define the hidden shift problem in terms of the action $\hat{*}$, so the decomposition of $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})/\ker(\pi)$ suffices. Since $\hat{*}$ factors through $*$ and π is a homomorphism, $([\mathfrak{b}_1]^{x_1} \cdots [\mathfrak{b}_k]^{x_k}) \mathbin{\hat{*}} j(E_0) = (\pi([\mathfrak{b}_1])^{x_1} \cdots \pi([\mathfrak{b}_k])^{x_k}) * j(E_0)$. Thus, by Lemma 5.1, $\hat{f}_0, \hat{f}_1$ hide $[\mathfrak{b}_1]^{s_1} \cdots [\mathfrak{b}_k]^{s_k}$, which shows that the output is correct. As in Theorem 5.3, Step 5 takes time $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$ using Theorem 5.1 and the modification of Algorithm 3 described above (or time $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2} + \sqrt{2})$ with Theorem 5.2). Finally, Step 6 is negligible.

Note that the output of Algorithm 5 is an element of $\mathrm{Cl}(\mathcal{O}_{\hat{\Delta}})$, not $\mathrm{Cl}(\mathcal{O}_\Delta)$, but it nevertheless defines an isogeny from E_0 to E_1.

Algorithm 6 Evaluating prime degree isogenies

Input: A discriminant $\Delta < 0$, an elliptic curve $E/\mathbb{F}_q$ with $\mathrm{End}(E) = \mathcal{O}_\Delta$, a point $P \in E(\mathbb{F}_q)$ such that $[\mathrm{End}(E) : \mathbb{Z}[\mathrm{Frob}_q]]$ and $\#E(\mathbb{F}_q)$ are coprime, and an $\mathrm{End}(E)$-ideal $\mathfrak{L} = (\ell, c + d\,\mathrm{Frob}_q)$ of prime norm $\ell \neq \mathrm{char}(\mathbb{F}_q)$ not dividing the index $[\mathrm{End}(E) : \mathbb{Z}[\mathrm{Frob}_q]]$

Output: The unique elliptic curve E' admitting a normalized isogeny $\phi\colon E \to E'$ with kernel $E[\mathfrak{L}]$, and the x-coordinate of $\phi(P)$ for $\Delta \neq -3, -4$ or the square (resp. cube) of the x-coordinate otherwise

1: Using Algorithm 2 with any valid choice of t, compute a relation $\mathbf{z} \in \mathbb{Z}^f$ such that $[\mathfrak{L}] = [\mathcal{F}^{\mathbf{z}}] = [\mathfrak{p}_1^{z_1}\mathfrak{p}_2^{z_2}\cdots\mathfrak{p}_f^{z_f}]$
2: Compute a sequence of isogenies $(\phi_1, \ldots, \phi_s)$ such that the composition $\phi_c\colon E \to E_c$ of the sequence has kernel $E[\mathfrak{p}_1^{z_1}\mathfrak{p}_2^{z_2}\cdots\mathfrak{p}_f^{z_f}]$, using the method of [4, §3]
3: Using Cornacchia's algorithm, find a generator $\alpha \in \mathcal{O}_\Delta$ of the fractional ideal $\mathfrak{L}/(\mathfrak{p}_1^{z_1}\mathfrak{p}_2^{z_2}\cdots\mathfrak{p}_f^{z_f})$
4: Evaluate $\phi_c(P) \in E_c(\mathbb{F}_q)$
5: Write $\alpha = (u + v\,\mathrm{Frob}_q)/z$, compute the isomorphism $\eta\colon E_c \xrightarrow{\sim} E'$ with $\eta^*(\omega_{E'}) = (u/z)\omega_{E_c}$, and compute $Q = \eta(\phi_c(P))$
6: Compute $z^{-1} \bmod \#E(\mathbb{F}_{q^n})$ and $R = (z^{-1}(u + v\,\mathrm{Frob}_q))(Q)$
7: Put $r = x(R)^{|\mathcal{O}_\Delta^*|/2}$ and return (E', r)

7 A classical algorithm for evaluating isogenies

Algorithm 3 from Section 4, which computes the action of $\mathrm{Cl}(\mathcal{O}_\Delta)$ on (isomorphism classes of) elliptic curves, extends readily to yield an algorithm for evaluating isogenies in subexponential time. Since every isogeny factors into a composition of isogenies of prime degree, we focus on the case of prime degree for simplicity.

Fix an elliptic curve E over $\mathbb{F}_q$. For each Elkies prime ℓ, there exist up to isomorphism at most two horizontal isogenies $\phi\colon E \to E'$ of degree ℓ. As in [4, 16], we distinguish between the two cases by specifying the exact ideal $\mathfrak{L} \subset \mathrm{End}(E) = \mathcal{O}_\Delta$ of norm ℓ corresponding to the kernel of ϕ. Likewise, as in [4, 16], to distinguish between two isogenies that are identical up to isomorphism, we impose the condition

$$\phi^*(w_{E'}) = w_E$$

where w_E denotes the invariant differential of E. The (unique) isogeny of kernel $\mathfrak{L}$ satisfying these conditions is said to be a *normalized* isogeny.

The procedure for evaluating ϕ given E and $\mathfrak{L}$ is outlined in Algorithm 6. This algorithm is identical to Algorithm 3 except that Steps 3–7 are new. Observe that Step 3 of Algorithm 6 is identical to Steps 20–23 of [16, Algorithm 3], and Steps 4–7 of Algorithm 6 are identical to Steps 4–7 of [16, Algorithm 4]. From the runtime analysis in [16, §4.4], we see that the running time of Steps 3–7 is dominated by the running time of Step 2. Hence Algorithm 6 also has an asymptotic running time of $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$. We remark that Algorithm 6 improves upon [16, Algorithm 4] in the sense that no unproven assumptions (other than GRH) are needed for proving its correctness or its running time bound.

8 Related work

We conclude by discussing the relationship of our results to prior work.

The question of how efficiently isogenies can be computed was first considered by Galbraith [11]. The central idea of speeding up an isogeny computation by reducing an ideal modulo principal ideals to obtain a smooth ideal is originally due to Galbraith, Hess, and Smart [12]. Their algorithm remains the best known classical algorithm for computing an isogeny between two given endomorphic

curves, with an exponential running time of $O(p^{1/4})$ under heuristic assumptions. No attempt is made in [12] to improve the running time to subexponential, in part because their main application (extending Weil descent) does not benefit from such optimization.

Bröker, Charles, and Lauter [4] use the same central idea of computing modulo ideal classes to give an algorithm for evaluating isogenies between endomorphic curves, given a kernel ideal. Their algorithm runs in polynomial time, but only for small discriminants. The algorithm of Jao and Soukharev [16] may be viewed as a generalization of [4] to large discriminants. It represents the first (heuristically) subexponential algorithm for evaluating isogenies of a given kernel ideal, with a heuristic running time of $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$. We remark that the aforementioned innovation in Galbraith et al. [12], of calculating modulo ideal classes, has found application in other related problems as well. For example, the algorithm of Bisson and Sutherland [3], discovered independently, uses the same idea to compute endomorphism rings in (heuristically) $L_q(\frac{1}{2}, \frac{\sqrt{3}}{2})$ time.

We stress that, with the exception of [4], which is restricted in scope to small discriminants, all the results mentioned above make heuristic assumptions of varying severity [3, §4][11, p. 126][12, p. 37][16, p. 224] in addition to the Generalized Riemann Hypothesis in the course of proving their respective runtime claims. Our work is the first to achieve provably subexponential running time with no heuristic assumptions other than GRH.

In practice, the heuristic algorithms in [3] and [16] run slightly faster than the algorithms described in Sections 4 and 7, because they make use of an optimized exponent distribution (originating from [3]) that minimizes the number of large degree isogenies appearing in ϕ_c. Our work does not use this optimization, because doing so would reintroduce the need for additional heuristic assumptions.

Lattice-based approaches

An alternative approach to computing isogenies, given in Couveignes [7, p. 11] and Stolbunov [32, p. 227], is to treat the class group as a $\mathbb{Z}$-module and use lattice basis reduction to compute the group action. In practice, the lattice-based approach works well for moderate parameter sizes. However, the method asymptotically requires exponential time (even with known quantum algorithms), and thus is slower than our approach.

The basic idea of the lattice-based approach is to choose a factor base $\mathcal{F}$ of small primes with the property that $\mathcal{F}$ generates $\mathrm{Cl}(\mathcal{O}_\Delta)$. We then have a surjective homomorphism $\psi\colon \mathbb{Z}^f \to \mathrm{Cl}(\mathcal{O}_\Delta)$ where $f = |\mathcal{F}|$. The kernel L of this homomorphism is a lattice in $\mathbb{Z}^f$. As a pre-computation, we find an LLL-reduced basis for L. The group $\mathbb{Z}^f/L$ is then isomorphic to $\mathrm{Cl}(\mathcal{O}_\Delta)$ via ψ. Given a group element $g \in \mathbb{Z}^f/L$, we can compute the action of g on $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ as follows. Solve the closest vector problem (CVP) on L to obtain a short vector g' congruent to $g \bmod L$, and then compute the action of g' on $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ directly. The running time of the latter step is directly proportional to the norm of g'.

The proposed method works relatively well in practice, taking 229 seconds to compute a specially selected 428-bit example on PC hardware [32, Table 1]. Nevertheless, it suffers from drawbacks that make it unsuitable for theoretical analysis. Under GRH, the dimension f of the lattice must be $6(\ln|\Delta|)^2$ in order to guarantee ψ is surjective [1]. Even under aggressive heuristic assumptions, a bound of $f \approx \ln|\Delta|$ is still necessary [32, p. 224]. The best available algorithms for solving CVP on lattices of this size require either exponential time to achieve a polynomial approximation factor, or polynomial time with an (almost) exponential approximation factor of $2^{f \ln\ln f/\ln f}$ [24]. Since the

computation of the action of g' on $\mathrm{Ell}_{q,n}(\mathcal{O}_\Delta)$ takes time proportional to the approximation factor, either choice yields a slower algorithm than ours.

Note that these approaches apparently do not benefit from known quantum algorithms. Although algorithms for the abelian hidden shift problem can be applied to find short vectors in lattices [23], the overhead involved means that when the result of Theorem 5.1 (or Theorem 5.2) is applied, the resulting algorithms are no faster than the best known classical algorithms. Additionally, this approach only applies to the unique shortest vector problem, and is not known to apply to CVP, which may be even harder.

Acknowledgments

This work was supported in part by MITACS, NSERC, QuantumWorks, and the US ARO/DTO.

References

1. Eric Bach. Explicit bounds for primality testing and related problems. *Math. Comp.*, 55(191):355–380, 1990.
2. Daniel J. Bernstein. How to find smooth parts of integers, 2004. `http://cr.yp.to/papers.html#smoothparts`.
3. Gaetan Bisson and Andrew V. Sutherland. Computing the endomorphism ring of an ordinary elliptic curve over a finite field. *J. Number Theory*, to appear, 2009.
4. Reinier Bröker, Denis Charles, and Kristin Lauter. Evaluating large degree isogenies and applications to pairing based cryptography. In *Pairing '08: Proceedings of the 2nd International Conference on Pairing-Based Cryptography*, pages 100–112, 2008.
5. Johannes Buchmann and Ulrich Vollmer. *Binary quadratic forms: An algorithmic approach*, volume 20 of *Algorithms and Computation in Mathematics*. Springer, Berlin, 2007.
6. Kevin K. H. Cheung and Michele Mosca. Decomposing finite abelian groups. *Quantum Inform. Comput.*, 1(3):26–32, 2001.
7. Jean-Marc Couveignes. Hard homogeneous spaces, 2006. `http://eprint.iacr.org/2006/291`.
8. David A. Cox. *Primes of the form $x^2 + ny^2$: Fermat, class field theory and complex multiplication*. Wiley, New York, 1989.
9. Wim van Dam, Sean Hallgren, and Lawrence Ip. Quantum algorithms for some hidden shift problems. In *SODA '02: Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 489–498, 2002.
10. Luca De Feo. Fast algorithms for computing isogenies between ordinary elliptic curves in small characteristic. *J. Number Theory*, to appear, 2010.
11. Steven D. Galbraith. Constructing isogenies between elliptic curves over finite fields. *LMS J. Comput. Math.*, 2:118–138 (electronic), 1999.
12. Steven D. Galbraith, Florian Hess, and Nigel P. Smart. Extending the GHS Weil descent attack. In *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Comput. Sci.*, pages 29–44, 2002.
13. Sean Hallgren. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. *J. ACM*, 54(1):article 4, 2007. Preliminary version in STOC '02.
14. Sorina Ionica and Antoine Joux. Pairing the volcano. In *Algorithmic number theory: Proceedings of ANTS-IX*, volume 6197 of *Lecture Notes in Comput. Sci.*, pages 201–218, 2010.
15. David Jao, Stephen D. Miller, and Ramarathnam Venkatesan. Expander graphs based on GRH with an application to elliptic curve cryptography. *J. Number Theory*, 129(6):1491–1504, 2009.
16. David Jao and Vladimir Soukharev. A subexponential algorithm for evaluating large degree isogenies. In *Algorithmic number theory: Proceedings of ANTS-IX*, volume 6197 of *Lecture Notes in Comput. Sci.*, pages 219–233, 2010.
17. Alexei Yu. Kitaev. Quantum measurements and the abelian stabilizer problem. arXiv:quant-ph/9511026.
18. Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM J. Comput.*, 35(1):170–188, 2005.
19. Reynald Lercier and Thomas Sirvent. On Elkies subgroups of l-torsion points in elliptic curves defined over a finite field. *J. Théor. Nombres Bordeaux*, 20(3):783–797, 2008.

20. Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. Inform. Theory*, 39(5):1639–1646, 1993.
21. Daniele Micciancio and Oded Regev. Lattice-based cryptography. In D. J. Bernstein, J. Buchmann, and E. Dahmen, editors, *Post-quantum cryptography*, pages 147–191. Springer, Berlin, 2009.
22. Oded Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. arXiv:quant-ph/0406151.
23. Oded Regev. Quantum computation and lattice problems. *SIAM J. Comput.*, 33(3):738–760, 2004.
24. Oded Regev. On the complexity of lattice problems with polynomial approximation factors. In Phong Q. Nguyen and Brigitte Valle, editors, *The LLL Algorithm*, Information Security and Cryptography, pages 475–496. Springer, Berlin, 2010.
25. Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies, 2006. `http://eprint.iacr.org/2006/145`.
26. Arnold Schönhage. Fast reduction and composition of binary quadratic forms. In *ISSAC '91: Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation*, pages 128–133, 1991.
27. René Schoof. Counting points on elliptic curves over finite fields. *J. Théor. Nombres Bordeaux*, 7(1):219–254, 1995. Les Dix-huitièmes Journées Arithmétiques (Bordeaux, 1993).
28. Jean-Pierre Serre. Groupes de Lie l-adiques attachés aux courbes elliptiques. In *Les Tendances Géom. en Algébre et Théorie des Nombres*, pages 239–256. Éditions du Centre National de la Recherche Scientifique, Paris, 1966.
29. Martin Seysen. A probabilistic factorization algorithm with quadratic forms of negative discriminant. *Math. Comp.*, 48(178):757–780, 1987.
30. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. Preliminary version in FOCS '94.
31. Joseph H. Silverman. *The arithmetic of elliptic curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1992. Corrected reprint of the 1986 original.
32. Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. Math. Commun.*, 4(2):215–235, 2010.
33. John Tate. Endomorphisms of abelian varieties over finite fields. *Invent. Math.*, 2:134–144, 1966.
34. Jacques Vélu. Isogénies entre courbes elliptiques. *C. R. Acad. Sci. Paris Sér. A-B*, 273:A238–A241, 1971.
35. William C. Waterhouse. Abelian varieties over finite fields. *Ann. Sci. École Norm. Sup. (4)*, 2:521–560, 1969.
36. John Watrous. Quantum algorithms for solvable groups. In *STOC '01: Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 60–67, 2001.

A Subexponential-time quantum algorithm for the general abelian hidden shift problem with polynomial space

Following Kuperberg's discovery of a subexponential-time quantum algorithm for the hidden shift problem in any finite abelian group A [18], Regev presented a modification of Kuperberg's algorithm that requires only polynomial space, with a slight increase in the running time [24]. However, Regev only explicitly considered the case $A = \mathbb{Z}_{2^n}$, and while he showed that the running time is $L_{|A|}(\frac{1}{2}, c)$, he did not determine the value of the constant c.

In this appendix we describe a polynomial-space quantum algorithm for the general abelian hidden shift problem using time $L_{|A|}(\frac{1}{2}, \sqrt{2})$. We use several of the same techniques employed by Kuperberg [18, Algorithm 5.1 and Thm. 7.1] to go beyond the case $A = \mathbb{Z}_{2^n}$, adapted to work with a Regev-style sieve that only uses polynomial space.

Let $A = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_t}$ be a finite abelian group. Consider the hidden shift problem with hidden shift $s = (s_1, \ldots, s_t) \in A$. By Fourier sampling, one (coherent) evaluation of the hiding functions f_0, f_1 is sufficient to produce the state

$$|\psi_x\rangle := \frac{1}{\sqrt{2}} \left(|0\rangle + \exp\left[2\pi i \left(\frac{s_1 x_1}{N_1} + \cdots + \frac{s_t x_t}{N_t} \right) \right] |1\rangle \right) \qquad (\psi)$$

with a known value $x = (x_1, \ldots, x_t) \in_{\mathrm{R}} A$ (see for example the proof of Theorem 7.1 in [18]), where $x \in_{\mathrm{R}} A$ denotes that x occurs uniformly at random from A. For simplicity, we begin by considering the case where $A = \mathbb{Z}_N$ is cyclic. Then Fourier sampling produces states

$$|\psi_x\rangle = \frac{1}{\sqrt{2}}(|0\rangle + \omega^{sx}|1\rangle)$$

where $x \in_{\mathrm{R}} \mathbb{Z}_N$ is known and $\omega := e^{2\pi i/N}$.

If we could make states $|\psi_x\rangle$ with chosen values of x, then we could determine s. In particular, the following observation is attributed to Peter Høyer in [18]:

Lemma A.1. *Given one copy each of the states $|\psi_1\rangle, |\psi_2\rangle, |\psi_4\rangle, \ldots, |\psi_{2^{k-1}}\rangle$, where $2^k = \Omega(N)$, one can reconstruct s in polynomial time with probability $\Omega(1)$.*

Proof. We have

$$\bigotimes_{j=0}^{k-1} |\psi_{2^j}\rangle = \frac{1}{\sqrt{2^k}} \sum_{y=0}^{2^k-1} \omega^{sy}|y\rangle.$$

Apply the inverse quantum Fourier transform over $\mathbb{Z}_N$ (which takes time $\mathrm{poly}(\log N)$ [17]) and measure in the computational basis. The Fourier transform of $|s\rangle$, namely $\frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega^{sy}|s\rangle$, has overlap squared with this state of $2^k/N$, which implies the claim.

We aim to produce states of the form $|\psi_{2^j}\rangle$ using a sieve that combines states to prepare new ones with more desirable labels. A basic building block is Algorithm 7, which can be used to produce states with smaller labels.

Lemma A.2. *Algorithm 7 runs in time $2^k \,\mathrm{poly}(\log N)$ and succeeds with probability $\Omega(1)$ provided $4k \le B/B' \le 2^k/k$.*

Proof. The running time is dominated by the brute force calculation in Step 6 and the projection in Step 10, both of which can be performed in time $2^k \,\mathrm{poly}(\log N)$.

The probability of aborting in Step 2 for any one x_i is $1 - \frac{2B'}{B}\lfloor \frac{B}{2B'} \rfloor \le \frac{2B'}{B}$, so by the union bound, the overall probability of aborting in this step is at most $k\frac{2B'}{B} \le 1/2$. Conditioned on not aborting in Step 2, $x_i \in_{\mathrm{R}} \{0, 1, \ldots, 2B'\lfloor B/2B' \rfloor - 1\}$.

Let $x \cdot y^j = q(2B') + r^j$ where $0 \le r^j < 2B'$ (q is the measurement outcome, which is independent of j). By the uniformity of the x_is, each $r^j = x \cdot y^j \bmod 2B'$ is uniformly distributed over $\{0, 1, \ldots, 2B' - 1\}$. Thus the output label is $x' = x \cdot (y^\star - y^\star) = |r^\star - r^\star|$ where $r^\star, r^\star \in_{\mathrm{R}} \{0, 1, \ldots, 2B' - 1\}$. A simple calculation shows that the distribution of $|r^\star - r^\star|$ is

$$\Pr(|r^\star - r^\star| = \Delta) = \begin{cases} \frac{1}{2B'} & \text{for } \Delta = 0 \\ \frac{2B'-\Delta}{2B'^2} & \text{for } \Delta \in \{1, \ldots, 2B' - 1\}. \end{cases}$$

Thus the probability that we abort in Steps 12–16 is $1/2$, and conditioned on not aborting in these steps, $x' \in_{\mathrm{R}} \{0, 1, \ldots, B' - 1\}$. Thus the algorithm is correct if it reaches Step 17.

It remains to show that the algorithm succeeds with constant probability. We have already bounded the probability that we abort in Step 2 and Steps 12–16. Since $y = 0$ occurs with probability 2^{-k} and at most one state $|y^\nu\rangle$ can be unpaired (and this only happens when ν is odd), the

Algorithm 7 Combining states to give smaller labels

Input: Parameters B, B' and states $|\psi_{x_1}\rangle, \ldots, |\psi_{x_k}\rangle$ with known $x_1, \ldots, x_k \in_{\mathrm{R}} \{0, 1, \ldots, B-1\}$
Output: State $|\psi_{x'}\rangle$ with known $x' \in_{\mathrm{R}} \{0, 1, \ldots, B'-1\}$
1: **if** $\exists i\colon x_i \geq 2B' \lfloor B/2B' \rfloor$ **then**
2: Abort
3: **end if**
4: Introduce an ancilla register and compute

$$\frac{1}{\sqrt{2^k}} \sum_{y \in \{0,1\}^k} \omega^{s(x \cdot y)} |y\rangle |\lfloor (x \cdot y)/2B' \rfloor\rangle$$

where $x \cdot y := \sum_{i=1}^k x_i y_i$
5: Measure the ancilla register, giving an outcome q and a state

$$\frac{1}{\sqrt{\nu}} \sum_{j=1}^{\nu} \omega^{s(x \cdot y^j)} |y^j\rangle$$

where $y^1, \ldots, y^\nu \neq 0^k$ are the k-bit strings such that $\lfloor (x \cdot y^j)/2B' \rfloor = q$
6: Compute $y^1, \ldots, y^\nu$ by brute force
7: **if** $\nu = 1$ **then**
8: Abort
9: **end if**
10: Project onto $\mathrm{span}\{|y^1\rangle, |y^2\rangle\}$ or $\mathrm{span}\{|y^3\rangle, |y^4\rangle\}$ or ... or $\mathrm{span}\{|y^{2\lfloor \nu/2 \rfloor - 1}\rangle, |y^{2\lfloor \nu/2 \rfloor}\rangle\}$, giving an outcome $\mathrm{span}\{|y^\star\rangle, |y^\star\rangle\}$
11: Let $x' = x \cdot (y^\star - y^\star)$ where $x \cdot y^\star \geq x \cdot y^\star$ WLOG
12: **if** $x' \in \{1, \ldots, B'-1\}$ **then**
13: Abort with probability $B'/(2B' - x')$
14: **else if** $x' \in \{B', \ldots, 2B'-1\}$ **then**
15: Abort
16: **end if**
17: Relabel $|y^\star\rangle \mapsto |0\rangle$ and $|y^\star\rangle \mapsto |1\rangle$, giving a state $|\psi_{x'}\rangle$

projection in Step 10 fails with probability at most $\nu^{-1} + 2^{-k} \leq 1/3 + o(1)$. We claim that the probability of aborting in Step 8 (i.e., the probability that $\nu = 1$) is also bounded away from 1. Call a value of q bad if $\nu = 1$. Since $0 \leq x \cdot y \leq k(B-1)$, there are at most $kB/2B'$ possible values of q, and in particular, there can be at most $kB/2B'$ bad values of q. Since the probability of any particular bad q is $1/2^k$, the probability that q is bad is at most $kB/B'2^{k+1} \leq 1/2$. This completes the proof.

We also use a combination procedure that zeros out low-order bits, as described in Algorithm 8. Note that we only use this in the case where N is a power of 2. We use the notation $xS := \{xz : z \in S\}$ for any $x \in \mathbb{Z}$ and $S \subset \mathbb{Z}$.

Lemma A.3. *Algorithm 8 runs in time* $2^k \operatorname{poly}(\log N)$ *and succeeds with probability* $\Omega(1)$ *provided* $k \geq \ell' - \ell + 1$.

Proof. The proof is similar to that of Lemma A.2. Again the running time is dominated by the brute force calculation in Step 3 and the projection in Step 7, both of which can be performed in time $2^k \operatorname{poly}(\log N)$.

Algorithm 8 Combining states to cancel low-order bits

Input: Parameters ℓ, ℓ' and states $|\psi_{x_1}\rangle, \ldots, |\psi_{x_k}\rangle$ with known $x_1, \ldots, x_k \in_{\mathrm{R}} 2^{\ell}\{0, 1, \ldots, N/2^{\ell} - 1\}$
Output: State $|\psi_{x'}\rangle$ with known $x' \in_{\mathrm{R}} 2^{\ell'}\{0, 1, \ldots, N/2^{\ell'} - 1\}$

1: Introduce an ancilla register and compute

$$\frac{1}{\sqrt{2^k}} \sum_{y \in \{0,1\}^k} \omega^{s(x \cdot y)} |y\rangle |x \cdot y \bmod 2^{\ell'}\rangle$$

2: Measure the ancilla register, giving an outcome r and a state

$$\frac{1}{\sqrt{\nu}} \sum_{j=1}^{\nu} \omega^{s(x \cdot y^j)} |y^j\rangle$$

where $y^1, \ldots, y^\nu \neq 0^k$ are the k-bit strings such that $x \cdot y^j \bmod 2^{\ell'} = r$
3: Compute $y^1, \ldots, y^\nu$ by brute force
4: **if** $\nu = 1$ **then**
5: Abort
6: **end if**
7: Project onto $\mathrm{span}\{|y^1\rangle, |y^2\rangle\}$ or $\mathrm{span}\{|y^3\rangle, |y^4\rangle\}$ or ... or $\mathrm{span}\{|y^{2\lfloor \nu/2 \rfloor - 1}\rangle, |y^{2\lfloor \nu/2 \rfloor}\rangle\}$, giving an outcome $\mathrm{span}\{|y^\star\rangle, |y^\star\rangle\}$
8: Relabel $|y^\star\rangle \mapsto |0\rangle$ and $|y^\star\rangle \mapsto |1\rangle$, giving a state $|\psi_{x'}\rangle$ with $x' = x \cdot (y^\star - y^\star) \bmod N$

Algorithm 9 Sieving quantum states

Input: Procedures to prepare states from a set S_0 and to combine k states from S_{i-1} to make a state from S_i with probability at least p for each $i \in \{1, \ldots, m\}$
Output: State from S_m

1: **repeat**
2: **while** for all i we have fewer than k states from S_i **do**
3: Make a state from S_0
4: **end while**
5: Combine k states from some S_i to make a state from S_{i+1} with probability at least p
6: **until** there is a state from S_m

We claim that the algorithm is correct if it reaches Step 8. Observe that $x \cdot y^j \bmod N = q^j 2^{\ell'} + r$ where r is independent of j. Since $y^j \neq 0^k$, $x \cdot y^j \bmod N \in_{\mathrm{R}} 2^{\ell}\{0, 1, \ldots, N/2^{\ell} - 1\}$, so $q^j \in_{\mathrm{R}} \{0, 1, \ldots, N/2^{\ell'} - 1\}$, and hence $x' = (q^\star - q^\star)2^{\ell'} \bmod N \in_{\mathrm{R}} 2^{\ell'}\{0, 1, \ldots, N/2^{\ell'} - 1\}$ as required.

The projection in Step 7 fails with probability at most $1/3 + o(1)$. It remains to show that the algorithm reaches Step 7 with probability $\Omega(1)$, i.e., to upper bound the probability that $\nu = 1$. Call a value of r bad if $\nu = 1$. There are $2^{\ell' - \ell}$ possible values of r, so in particular there are at most $2^{\ell' - \ell}$ bad values of r. Since the probability of any particular bad r is $1/2^k$, the probability that r is bad is at most $2^{\ell' - \ell - k} \leq 1/2$. This completes the proof.

Algorithm 8 differs from the analogous procedure in [22] in that the latter requires $\nu = O(1)$, which is established in the analysis using a second moment argument. The modification of pairing as many values of y as possible allows us to use a simpler analysis (with essentially the same performance).

We apply these combination procedures in the generalized sieve of Algorithm 9, which is equivalent to Regev's "pipeline of routines" [22].

Lemma A.4. *Suppose $me^{-2k} = o(1)$. Then Algorithm 9 is correct, succeeds with probability $1-o(1)$ using $k^{(1+o(1))m}$ state preparations and combination operations, and uses space $O(mk)$.*

Proof. If Algorithm 9 outputs a state from S_m then it is correct. Since the algorithm never stores more than $O(mk)$ states at a time, it uses space $O(mk)$. It remains to show that the algorithm is likely to succeed using only $k^{(1+o(1))m}$ state preparations and combination operations.

If we could perform combinations deterministically, we would need

$$\begin{aligned} 1 \text{ state from } S_m, \\ k \text{ states from } S_{m-1}, \\ k^2 \text{ states from } S_{m-2}, \\ \vdots \\ k^m \text{ states from } S_0. \end{aligned}$$

Since the combinations only succeed with probability p, we lower bound the probability of eventually producing $(2k/p)^{m-i}$ states from S_i for each $i \in \{1, \ldots, m\}$ (so in particular, we produce one state from S_m). Given $(2k/p)^{m-i+1}$ states from S_{i-1}, the expected number of successful combinations is $p(2k/p)^{m-i+1}/k = 2(2k/p)^{m-i}$, whereas only $(2k/p)^{m-i}$ successful combinations are needed. By the Chernoff bound, the probability of having fewer than $(2k/p)^{m-i}$ successful combinations is at most $e^{-p(2k/p)^{m-i}}$. Thus, by the union bound, the probability that the algorithm fails is at most

$$\sum_{i=1}^{m-1} e^{-p(2k/p)^{m-i}} \leq me^{-2k},$$

so the probability of success is $1 - o(1)$.

Finally, the number of states from S_0 is $(2k/p)^m = k^{(1+o(1))m}$ and the total number of combinations is $\sum_{i=0}^{m-1}(2k/p)^{m-i}/k = k^{(1+o(1))m}$.

When using the sieve, we have the freedom to choose the relationship between k and m to optimize the running time. Suppose that $mk = (1 + o(1)) \log_2 N$ (intuitively, to cancel $\log_2 N$ bits of the label), and also suppose that the combination operation takes time $2^k \operatorname{poly}(\log N)$ (as in Lemma A.2 and Lemma A.3). Then if we take $k = c\sqrt{\log_2 N \log_2 \log_2 N}$, we find that the overall running time of Algorithm 9 is $2^k 2^{(1+o(1))m \log_2 k} \operatorname{poly}(\log N) = L_N(\frac{1}{2}, c + \frac{1}{2c})$. Choosing $c = \frac{1}{\sqrt{2}}$ gives the best running time, $L_N(\frac{1}{2}, \sqrt{2})$.

We now consider how to apply the sieve. To use Lemma A.1, our goal is to prepare states of the form $|\psi_{2^j}\rangle$ for $j \in \{0, 1, \ldots, \lfloor \log_2 N \rfloor\}$. First we show how to prepare the state $|\psi_1\rangle$ in time $L_N(\frac{1}{2}, \sqrt{2})$ using Algorithm 7 as the combination procedure in Algorithm 9. For $i \in \{0, 1, \ldots, m\}$, the ith stage of the sieve produces states with labels from $S_i = \{0, 1, \ldots, B_i - 1\}$. Lemma A.5 below shows that there is a choice of the B_i with $B_0 = N$, $B_m = 2$, and successive ratios of the B_is satisfying the conditions of Lemma A.2, such that $2^k k^{(1+o(1))m} = L_N(\frac{1}{2}, \sqrt{2})$. It then follows that Algorithm 9 produces a uniformly random label from $S_m = \{0, 1\}$ with constant probability in time $L_N(\frac{1}{2}, \sqrt{2})$, and in particular can be used to produce a copy of $|\psi_1\rangle$ in time $L_N(\frac{1}{2}, \sqrt{2})$.

Lemma A.5. *There is a constant N_0 such that for all $N > N_0$, letting $B_i = \lfloor N/\rho^i \rfloor$ where $\rho = (N/2)^{1/m}$ and*

$$k = \left\lfloor \sqrt{\tfrac{1}{2} \log_2 N \log_2 \log_2 N} \right\rfloor \qquad m = \left\lceil \frac{\log_2 N/2}{k - \log_2 2k} \right\rceil = \Theta\left(\sqrt{\frac{\log_2 N}{\log_2 \log_2 N}} \right),$$

we have $B_0 = N$, $B_m = 2$, and $4k \leq B_{i-1}/B_i \leq 2^k/k$ for all $i \in \{1, \ldots, m\}$.

Proof. Clearly $B_0 = N$, and the value of ρ is chosen so that $B_m = 2$.

For $i \in \{1, \ldots, m\}$, we have

$$\frac{B_{i-1}}{B_i} = \frac{\lfloor N/\rho^{i-1} \rfloor}{\lfloor N/\rho^i \rfloor} \leq \frac{N/\rho^{i-1}}{N/\rho^i - 1} = \frac{\rho}{1 - \rho^i/N}.$$

Since $\rho^i/N \leq \rho^m/N = 1/2$, we have $B_{i-1}/B_i \leq 2\rho$. Then using

$$\rho \leq (N/2)^{\frac{k - \log_2 2k}{\log_2 N/2}} = \frac{2^k}{2k}$$

gives $B_{i-1}/B_i \leq 2^k/k$ as claimed.

Similarly, we have

$$\frac{B_{i-1}}{B_i} = \frac{\lfloor N/\rho^{i-1} \rfloor}{\lfloor N/\rho^i \rfloor} \geq \frac{N/\rho^{i-1} - 1}{N/\rho^i} = \rho - \rho^i/N \geq \rho - \tfrac{1}{2}.$$

Since

$$\rho = (N/2)^{\Theta(\sqrt{\log_2 \log_2 N / \log_2 N})} = 2^{\Theta(\sqrt{\log_2 N \log_2 \log_2 N})} = 2^{\Theta(k)},$$

we have $\rho - \frac{1}{2} \geq 4k$ for sufficiently large N. This completes the proof.

If N is odd, then division by 2 is an automorphism of $\mathbb{Z}_N$. Thus we can prepare $|\psi_{2^j}\rangle$ by performing the above sieve under the automorphism $x \mapsto 2^{-j}x$. It follows that the abelian hidden shift problem in a cyclic group of odd order N can be solved in time $L_N(\frac{1}{2}, \sqrt{2})$.

Now suppose that $N = 2^n$ is a power of 2. In this case, we first use Algorithm 8 to cancel low-order bits and then use Algorithm 7 to cancel high-order bits. Note that if all states $|\psi_x\rangle$ have labels x with a common factor—say, $2^j | x$—then we can view the labels as elements of $\mathbb{Z}_{2^{n-j}}$ and apply Algorithm 7 to affect the $n - j$ most significant bits. Specifically, to make the state $|\psi_{2^j}\rangle$, we apply Algorithm 9 using Algorithm 8 as the combination procedure that produces states from S_i using states from S_{i-1} for $i \in \{1, \ldots, m_1 + 1\}$, and Algorithm 7 (on the $n - j$ most significant bits) as the combination procedure for $i \in \{m_1 + 2, \ldots, m_1 + m_2 + 1\}$, taking

$$S_i = \begin{cases} 2^{(k-1)i}\{0, 1, \ldots, 2^{n-(k-1)i} - 1\} & \text{for } i \in \{0, 1, \ldots, m_1\} \\ 2^j\{0, 1, \ldots, B_i - 1\} & \text{for } i \in \{m_1 + 1, \ldots, m_1 + m_2 + 1\} \end{cases}$$

where now

$$B_i = \lfloor 2^{n-j}/\rho^{i-m_1-1} \rfloor \qquad m_1 = \lfloor j/(k-1) \rfloor$$
$$\rho = 2^{(n-j-1)/m_2} \qquad m_2 = \left\lceil \frac{n-j}{k - \log_2 2k} \right\rceil$$

and again $k = \lfloor\sqrt{\frac{1}{2}\log_2 N \log_2\log_2 N}\rfloor$. When making states in S_i from states in S_{i-1} for $i \in \{1,\ldots,m_1\}$, we cancel $k-1$ bits with k states, so the condition of Lemma A.3 is satisfied. For $i = m_1+1$, we cancel $j-(k-1)m_1 = j-(k-1)\lfloor j/(k-1)\rfloor \le j-(k-1)[j/(k-1)-1] = k-1$ bits, so again the condition of Lemma A.3 is satisfied. For $i \in \{m_1+2,\ldots,m_1+m_2+1\}$, Lemma A.5 implies that the conditions of Lemma A.2 are satisfied provided $2^{n-j} \ge N_0$. (If $2^{n-j} < N_0$ then we only need to perform the first m_1+1 stages of the sieve, producing a state uniformly at random from S_{m_1+1}; in this case $|S_{m_1+1}| = O(1)$, so $O(1)$ repetitions suffice to produce a copy of $|\psi_{2^j}\rangle$.) Finally, since $(m_1+m_2+1)k = (1+o(1))n$, the discussion following Lemma A.4 shows that Algorithm 9 takes time $L_N(\frac{1}{2},\sqrt{2})$.

So far we have covered the case where the group is $A = \mathbb{Z}_N$ with N either odd or a power of 2. Now consider the case of a general finite abelian group $A = \mathbb{Z}_{N_1}\times\cdots\times\mathbb{Z}_{N_t}$. By the Chinese remainder theorem, we can assume without loss of generality that each N_i is either odd or a power of 2. Consider what happens if we apply Algorithm 7 or Algorithm 8 to one component of a product of cyclic groups. Suppose we combine k states of the form of Eqn. (ψ). For each $i \in \{1,\ldots,k\}$, let $x_i \in \mathbb{Z}_{N_1}\times\cdots\times\mathbb{Z}_{N_t}$ denote the label of the ith state, with $x_{i,j} \in \mathbb{Z}_{N_j}$ for $j \in \{1,\ldots,t\}$. To address the ℓth component of A, the combination procedure prepares a state

$$\frac{1}{\sqrt{2^k}}\sum_{y\in\{0,1\}^k}\exp\left(2\pi i\sum_{i=1}^{k}\sum_{j=1}^{t}\frac{y_i x_{i,j} s_j}{N_j}\right)|y\rangle|h(\textstyle\sum_{i=1}^k x_{i,\ell}y_i)\rangle$$

for some function h (a quotient in Algorithm 7 or a remainder in Algorithm 8). For $j \ne \ell$, if $x_{i,j} = 0$ for all $i \in \{1,\ldots,k\}$ then $x'_j = \sum_{i=1}^k x_{i,j}(y_i^\star - y_i^\star) = 0$, so components that are initially zero remain zero. Thus, if we can prepare states $|\psi_x\rangle$ with $x_\ell \in_{\mathrm{R}} \mathbb{Z}_{N_\ell}$ and all other components zero, we effectively reduce the problem to the cyclic case.

To prepare such states, we use a new combination procedure, Algorithm 10. Without loss of generality, our goal is to zero out the first $t-1$ components, leaving the last one uniformly random from $\mathbb{Z}_{N_t}$. Algorithm 10 is similar to Algorithm 7, viewing the first $t-1$ components of the label $x_i \in \mathbb{Z}_{N_1}\times\cdots\times\mathbb{Z}_{N_t}$ as the mixed-radix integer

$$\mu(x_i) := \sum_{j=1}^{t-1} x_{i,j}\prod_{j'=1}^{j-1} N_{j'}.$$

Because we are merely trying to zero out certain components, we no longer require uniformity of the states output by the sieve, which simplifies the procedure and its analysis.

Lemma A.6. *Algorithm 10 runs in time* $2^k\,\mathrm{poly}(\log N)$ *and succeeds with probability* $\Omega(1)$ *provided* $B/B' \le 2^k/2k$.

Proof. As in Lemma A.2 and Lemma A.3, the running time is dominated by the brute force calculation in Step 3 and the projection in Step 7, both of which can be performed in time $2^k\,\mathrm{poly}(\log N)$.

We claim that the algorithm is correct if it reaches Step 8. Since $\sum_{i=1}^k \mu(x_i)y_i^j = qB' + r^j$ where q is independent of j and $0 \le r^j < B'$, we have

$$\mu(x') = \sum_{j=1}^{t-1} x'_j\prod_{j'=1}^{j-1} N_{j'} = \sum_{j=1}^{t-1}\sum_{i=1}^{k} x_{i,j}(y_i^\star - y_i^\star)\prod_{j'=1}^{j-1} N_{j'} = \sum_{i=1}^{k}\mu(x_i)(y_i^\star - y_i^\star) = r^\star - r^\star < B'$$

Algorithm 10 Combining non-cyclic states to reduce undesired components

Input: Parameters B, B' and states $|\psi_{x_1}\rangle, \ldots, |\psi_{x_k}\rangle$ with known $x_1, \ldots, x_k \in \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_t}$ satisfying $\mu(x_i) \in \{0, 1, \ldots, B-1\}$ for each $i \in \{1, \ldots, k\}$, with $x_{i,t} \in_{\mathrm{R}} \mathbb{Z}_{N_t}$

Output: State $|\psi_{x'}\rangle$ with known $x' \in \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_t}$ satisfying $\mu(x') \in \{0, 1, \ldots, B'-1\}$, with $x'_t \in_{\mathrm{R}} \mathbb{Z}_{N_t}$

1: Introduce an ancilla register and compute

$$\frac{1}{\sqrt{2^k}} \sum_{y \in \{0,1\}^k} \exp\left(2\pi i \sum_{i=1}^{k} \sum_{j=1}^{t} \frac{y_i x_{i,j} s_j}{N_j}\right) |y\rangle |\lfloor \textstyle\sum_{i=1}^{k} \mu(x_i) y_i / B' \rfloor\rangle$$

2: Measure the ancilla register, giving an outcome q and a state

$$\frac{1}{\sqrt{\nu}} \sum_{j=1}^{\nu} \exp\left(2\pi i \sum_{i=1}^{k} \sum_{j=1}^{t} \frac{y_i x_{i,j} s_j}{N_j}\right) |y^j\rangle$$

where $y^1, \ldots, y^\nu \neq 0^k$ are the k-bit strings such that $\lfloor \sum_{i=1}^{k} \mu(x_i) y_i^j / B' \rfloor = q$

3: Compute $y^1, \ldots, y^\nu$ by brute force

4: **if** $\nu = 1$ **then**

5: Abort

6: **end if**

7: Project onto $\mathrm{span}\{|y^1\rangle, |y^2\rangle\}$ or $\mathrm{span}\{|y^3\rangle, |y^4\rangle\}$ or ... or $\mathrm{span}\{|y^{2\lfloor \nu/2 \rfloor - 1}\rangle, |y^{2\lfloor \nu/2 \rfloor}\rangle\}$, giving an outcome $\mathrm{span}\{|y^\star\rangle, |y^\star\rangle\}$

8: Relabel $|y^\star\rangle \mapsto |0\rangle$ and $|y^\star\rangle \mapsto |1\rangle$ where $\sum_{i=1}^{k} \mu(x_i) y_i^\star \geq \sum_{i=1}^{k} \mu(x_i) y_i^\star$ WLOG, giving a state $|\psi_{x'}\rangle$ with $x'_j = \sum_{i=1}^{k} x_{i,j}(y_i^\star - y_i^\star)$ for each $j \in \{1, \ldots, t\}$

as required. Since $y^\star \neq y^\star$ and the $x_{i,t}$ are uniformly random, $x'_t = \sum_{i=1}^{k} x_{i,t}(y_i^\star - y_i^\star)$ is uniformly random as required.

The projection in Step 7 fails with probability at most $1/3 + o(1)$. We claim the algorithm reaches Step 7 with probability $\Omega(1)$. To show this, we need to upper bound the probability that $\nu = 1$. Call a value of q bad if $\nu = 1$. Since $0 \leq \sum_{i=1}^{k} \mu(x_i) y_i \leq k(B-1)$, there are at most kB/B' possible values of q, and in particular, there can be at most kB/B' bad values of q. Since the probability of any particular bad q is $1/2^k$, the probability that q is bad is at most $kB/B'2^k \leq 1/2$. This completes the proof.

To apply Algorithm 10 as the combination procedure for Algorithm 9, we require a straightforward variant of Lemma A.5, as follows.

Lemma A.7. *There is a constant N'_0 such that for all $N > N'_0$, letting $B_i = \lfloor N/\rho^i \rfloor$ where $\rho = N^{1/m}$ and*

$$k = \left\lfloor \sqrt{\tfrac{1}{2} \log_2 N \log_2 \log_2 N} \right\rfloor \qquad m = \left\lceil \frac{\log_2 N}{k - \log_2 4k} \right\rceil = \Theta\left(\sqrt{\frac{\log_2 N}{\log_2 \log_2 N}}\right),$$

we have $B_0 = N$, $B_m = 1$, and $B_{i-1}/B_i \leq 2^k/2k$ for all $i \in \{1, \ldots, m\}$.

Proof. Clearly $B_0 = N$, and the value of ρ is chosen so that $B_m = 1$.

We have $B_{m-1} \leq N/\rho^{m-1} = \rho$, and since

$$\rho \leq N^{\frac{k - \log_2 4k}{\log_2 N}} = \frac{2^k}{4k},$$

Algorithm 11 Abelian hidden shift problem

Input: Black box for the hidden shift problem in an abelian group A
Output: Hidden shift s

1: Write $A = \mathbb{Z}_{N_1} \times \cdots \times \mathbb{Z}_{N_t}$ where each N_i is either odd or a power of 2
2: **for all** $i \in \{1, \ldots, t\}$ **do**
3: **if** N_i is odd **then**
4: **for all** $j \in \{0, \ldots, \lfloor \log_2 N_i \rfloor\}$ **do**
5: Apply Algorithm 9, first using Algorithm 10 to zero out all components except the ith one and then using Algorithm 7 under the $\mathbb{Z}_{N_i}$-automorphism $x \mapsto 2^{-j}x$ to produce a copy of $|\psi_{(0,\ldots,0,2^j,0,\ldots,0)}\rangle$ (see the proof of Theorem A.1 for detailed parameters)
6: **end for**
7: **else**
8: Let $N_i = 2^n$
9: **for all** $j \in \{0, \ldots, n-1\}$ **do**
10: Apply Algorithm 9, first using Algorithm 10 to zero out all components except the ith one, then using Algorithm 8 to make states $|\psi_{(0,\ldots,0,x,0,\ldots,0)}\rangle$ with $2^j|x$, and finally using Algorithm 7 to produce a copy of $|\psi_{(0,\ldots,0,2^j,0,\ldots,0)}\rangle$ (see the proof of Theorem A.1 for detailed parameters)
11: **end for**
12: **end if**
13: Apply Lemma A.1 with $N = N_i$ to give s_i
14: **end for**
15: Output $s = (s_1, \ldots, s_t)$

the claimed inequality holds for $i = m$.

For $i \in \{1, \ldots, m-1\}$,

$$\frac{B_{i-1}}{B_i} = \frac{\lfloor N/\rho^{i-1} \rfloor}{\lfloor N/\rho^i \rfloor} \leq \frac{N/\rho^{i-1}}{N/\rho^i - 1} = \frac{\rho}{1 - \rho^i/N}.$$

Since $\rho^i/N \leq \rho^{m-1}/N = 1/\rho$, we have $B_{i-1}/B_i \leq \rho/(1 - 1/\rho)$. Then using

$$\rho = N^{\Theta(\sqrt{\log_2 \log_2 N/\log_2 N})} = 2^{\Theta(\sqrt{\log_2 N \log_2 \log_2 N})},$$

we have $\rho \geq 2$ provided $N > N_0'$ for some constant N_0', which implies $B_{i-1}/B_i \leq 2^k/2k$. This completes the proof.

Combining these ideas, the overall procedure is presented in Algorithm 11.

Theorem A.1. *Algorithm 11 runs in time* $L_{|A|}(\frac{1}{2}, \sqrt{2})$.

Proof. In Step 1, if the structure of the group is not initially known, it can be determined in polynomial time using [6]. Given the structure of the group, for each term $\mathbb{Z}_N$ we can easily factor $N = 2^n M$ where M is odd; then $\mathbb{Z}_N \cong \mathbb{Z}_{2^n} \times \mathbb{Z}_M$, and we obtain a decomposition of the desired form.

Now suppose without loss of generality that we are trying to determine s_t (i.e., $i = t$ in Step 2). The main contribution to the running time comes from the sieves in Step 5 (for N_t odd) and Step 10 (for N_t a power of 2).

First suppose that N_t is odd. It suffices to handle the case where $j = 0$, so we are making the state $|\psi_{(0,\ldots,0,1)}\rangle$. Then we apply Algorithm 9 with

$$S_i = \begin{cases} \{x \in A : \mu(x) < B_i\} & \text{for } i \in \{0, 1, \ldots, m_1\} \\ \{x \in A : \mu(x) = 0 \text{ and } x_t < B_i\} & \text{for } i \in \{m_1 + 1, \ldots, m_2\} \end{cases}$$

where

$$B_i = \begin{cases} \lfloor (N/N_t)/\rho_1^i \rfloor & \text{for } i \in \{0, 1, \ldots, m_1\} \\ \lfloor N_t/\rho_2^{i-m_1} \rfloor & \text{for } i \in \{m_1+1, \ldots, m_1+m_2\} \end{cases}$$

with

$$\rho_1 = (N/N_t)^{1/m_1} \qquad m_1 = \left\lceil \frac{\log_2 N/N_t}{k - \log_2 4k} \right\rceil$$

$$\rho_2 = (N_t/2)^{1/m_2} \qquad m_2 = \left\lceil \frac{\log_2 N_t/2}{k - \log_2 2k} \right\rceil$$

and $k = \lfloor \sqrt{\frac{1}{2} \log_2 N \log_2 \log_2 N} \rfloor$. We use Algorithm 10 as the combination procedure for the first m_1 stages of Algorithm 9. By Lemma A.7, the condition of Lemma A.6 is satisfied provided $N/N_t > N_0'$; otherwise we can produce a state with a label from S_{m_1} in only $O(1)$ trials. Then we proceed to apply Algorithm 7 as the combination procedure for the remaining m_2 stages of Algorithm 9. By Lemma A.5, the conditions of Lemma A.2 are satisfied provided $N_t > N_0$; otherwise, producing states with labels from S_{m_1} already suffices to produce the desired state with constant probability. Since $(m_1 + m_2)k = (1 + o(1)) \log_2 N$, Step 5 takes time $L_{|A|}(\frac{1}{2}, \sqrt{2})$ (see the discussion following the proof of Lemma A.4).

Now suppose that $N_t = 2^n$ is a power of 2. Then we apply Algorithm 9 with

$$S_i = \begin{cases} \{x \in A : \mu(x) < B_i\} & \text{for } i \in \{0, 1, \ldots, m_1\} \\ \{x \in A : \mu(x) = 0 \text{ and } x_t \in 2^{(k-1)i}\{0, 1, \ldots, 2^{n-(k-1)i}\}\} & \text{for } i \in \{m_1+1, \ldots, m_1+m_2\} \\ \{x \in A : \mu(x) = 0 \text{ and } x_t \in 2^j\{0, 1, \ldots, B_i - 1\}\} & \text{for } i \in \{m_1+m_2+1, \ldots, \\ & \qquad m_1+m_2+m_3+1\} \end{cases}$$

where

$$B_i = \begin{cases} \lfloor (N/N_t)/\rho_1^i \rfloor & \text{for } i \in \{0, 1, \ldots, m_1\} \\ \lfloor 2^{n-j}/\rho_3^{i-m_1-m_2-1} \rfloor & \text{for } i \in \{m_1+m_2+1, \ldots, m_1+m_2+m_3+1\} \end{cases}$$

with

$$\rho_1 = (N/N_t)^{1/m_1} \qquad m_1 = \left\lceil \frac{\log_2 N/N_t}{k - \log_2 4k} \right\rceil$$

$$m_2 = \lfloor j/(k-1) \rfloor$$

$$\rho_3 = 2^{(n-j-1)/m_3} \qquad m_3 = \left\lceil \frac{n-j-1}{k - \log_2 2k} \right\rceil$$

and again $k = \lfloor \sqrt{\frac{1}{2} \log_2 N \log_2 \log_2 N} \rfloor$. We use Algorithm 10 as the combination procedure for the first m_1 stages, Algorithm 8 for the next m_2+1 stages, and Algorithm 7 (on the $n-j$ most significant bits) for the final m_3 stages. By Lemma A.7 and Lemma A.5, the conditions of Lemma A.6 and Lemma A.2 are satisfied, respectively. Since we cancel at most $k-1$ bits in each stage that uses Algorithm 8, the conditions of Lemma A.3 are satisfied for the intermediate stages. Finally, since $(m_1 + m_2 + m_3 + 1)k = (1 + o(1)) \log_2 N$, Step 10 takes time $L_{|A|}(\frac{1}{2}, \sqrt{2})$.

The loops in Step 2, Step 4, and Step 9 only introduce polynomial overhead. Step 13 takes polynomial time and Step 15 is negligible. Thus the overall running time is $L_{|A|}(\frac{1}{2}, \sqrt{2})$ as claimed.