A subexponential-time, polynomial quantum space algorithm for inverting the CM group action

https://doi.org/10.1515/jmc-2015-0057
Received Feb 05, 2020; accepted Feb 14, 2020

Abstract: We present a quantum algorithm which computes group action inverses of the complex multiplication group action on isogenous ordinary elliptic curves, using subexponential time, but only polynomial quantum space. One application of this algorithm is that it can be used to find the private key from the public key in the isogeny-based CRS and CSIDH cryptosystems. Prior claims by Childs, Jao, and Soukharev of such a polynomial quantum space algorithm for this problem are false; our algorithm (along with contemporaneous, independent work by Biasse, Iezzi, and Jacobson) is the first such result.

Keywords: Isogeny-based cryptography, Quantum algorithms, Quantum cryptanalysis

2010 Mathematics Subject Classification: 68Q12, 94A60, 14K02

1 Introduction

In recent years, isogeny-based cryptosystems have emerged as a possible candidate for post-quantum cryptography. The earliest isogeny-based key agreement protocol, first proposed by Couveignes [9] and later by Rostovtsev and Stolbunov [18], uses the complex multiplication action of an imaginary quadratic ideal class group cl(\mathcal{O}) on an ordinary elliptic curve \( E(\mathbb{F}_q) \); we refer to this scheme as CRS. Very recently, a new isogeny-based proposal called CSIDH [4] has appeared, which is essentially equivalent to CRS except that it uses supersingular elliptic curves, and offers much faster performance. We emphasize that, in terms of security analysis, CRS and CSIDH are completely different from the supersingular case, which was first proposed for use in cryptography by Charles, Goren, and Lauter [5].

Both CRS and CSIDH can be broken (in the sense of a total break — recovery of the private key from the public key) by solving the group action inverse problem [20] on the complex multiplication group action, where the group in question is cl(\mathcal{O}). The first published quantum algorithm implementing this attack is the CJS algorithm [7], which breaks CRS and CSIDH in quantum subexponential running time, specifically in \( L_p(\frac{1}{2}, \frac{1}{\sqrt{2}}) \) operations.

The CJS attack also requires a subexponential amount of space, and of greater concern is that the quantum part of the algorithm also requires this much space. Since the history of this topic is convoluted, we present it in detail. The CJS attack consists of two parts: a classical algorithm (subexponential in time and space) to evaluate the complex multiplication action, and a quantum algorithm by Kuperberg to solve the di-
hedral hidden subgroup problem. The original version of Kuperberg’s quantum algorithm [13] required both subexponential time and subexponential space. Later improvements by Regev [17] and Kuperberg [14] reduced the space requirement to polynomial space. Based on these improvements, CJS [7, Remark 4.6] claimed that the CJS attack could be performed using polynomial quantum space. However, as pointed out by Galbraith and Vergnaud [11, §7.0.1], this claim is incorrect, because the classical subexponential algorithm to evaluate the complex multiplication action must be run in quantum superposition, and hence requires subexponential quantum space. The CJS authors have acknowledged this error in an errata on arXiv [6].

In this paper, we present a new algorithm for this problem which really requires only polynomial quantum space. Our algorithm uses a purely classical precomputation costing subexponential time and subexponential space, in which subexponentially slow lattice reduction algorithms are used to obtain a (polynomially long) advice string allowing any element of the class group to be expressed explicitly as a product of polynomially large primes with subexponentially large exponents. This precomputation then allows the complex multiplication action of \( \text{cl}(O) \) to be computed (both classically and quantumly) using subexponential time but only polynomial space, which is the ingredient needed for a polynomial quantum space group action inverse algorithm.

Couveignes [9], Stolbunov [20], and subsequent authors [10] have all specifically mentioned the use of LLL [15] for the purpose of evaluating the CM action. However, the use of LLL results in a fully exponential approximation factor, which is not good enough for our application. Our results are based on a variant of BKW [2] instead of LLL, and constitute (along with [1]; see below) the first detailed description of how to evaluate the complex multiplication operator in quantum subexponential time using only polynomial quantum space.

1.1 Related work

Recent work by Bonnetain and Schrottenloher [3] makes a detailed analysis of the security of CSIDH, by assessing the effectiveness of using lattice reduction algorithms, in particular BKZ [19], for the evaluation of the action of the class group. Their emphasis is on practical attacks rather than theoretical analysis.

Independent work of Biasse et al. [1] describes a quantum algorithm to evaluate the action, also requiring subexponential time and polynomial quantum space. Moreover, [1] presents a further time-space tradeoff to evaluate the action using only polynomial classical space. Compared to their work, our work provides a more self-contained description of the quantum processing steps.

2 Preliminaries

For an imaginary quadratic order \( O \), denote by \( \mathcal{E}_\mathbb{F}_p(O) \) the set of isomorphism classes of elliptic curves defined over \( \mathbb{F}_p \) with endomorphism ring isomorphic to \( O \). Let \( E \) be a curve in \( \mathcal{E}_\mathbb{F}_p(O) \). We have \( O \otimes \mathbb{Z} \mathbb{Q} = \mathbb{Q}(\sqrt{\Delta}) = \mathbb{K} \) where \( \Delta = \text{disc}(O) \). Denote the ideal class group of \( O \) by \( \text{cl}(O) \). For an ideal \( a \), we use \([a]\) to denote the equivalence class of \( a \) in \( \text{cl}(O) \). We may omit the \([\ ]\) when the meaning is clear from context.

The explicit group structure and even the size of the ideal class group can be difficult to compute classically. However, heuristically we can assume \( |\text{cl}(O)| = \sqrt{\Delta} \) for elliptic curves over \( \mathbb{F}_p \). When \( \Delta \) is a non-zero square modulo a prime \( \ell \), then the ideal \( \ell O \) splits into a product of conjugates \( \ell \). Such primes are called Elkies primes and we can explicitly write \( \ell O \) as \( \langle \ell, \pi - \lambda \rangle \) where \( \lambda \) is an eigenvalue of the Frobenius map acting on the \( \ell \)-torsion subgroup \( E[\ell] \) of \( E \). Set \( E[a] = \{(P \in E : \forall \psi \in a, \, \psi(P) = \infty \} \). When \( O \) is an imaginary quadratic order and \( \mathcal{E}_\mathbb{F}_p(O) \) is non-empty, there is a group action by \( \text{cl}(O) \) called complex multiplication via

\[
\text{cl}(O) \times \mathcal{E}_\mathbb{F}_p(O) \to \mathcal{E}_\mathbb{F}_p(O) \quad \quad ([a], E) \mapsto a \ast E = E/E[a],
\]

There is more than one way to evaluate the action of the ideal class group in practice, and most methods use some form of randomness. Since we will be implementing this action on a quantum computer, we choose...
one in particular [4, §8, Algorithm 2] which can most easily be modified to set randomness beforehand. Note that the ideals \( l_i = (\ell_i, \pi - 1) \) have kernel in \( \mathbb{F}_p \) while the kernel of \( \tilde{l}_i = (\ell_i, \pi + 1) \) is typically defined over \( \mathbb{F}_p \) but has some elements lying in \( \mathbb{F}_p / \mathbb{F}_p \).

Given two probability density functions \( \chi_1, \chi_2 \) over a finite set \( X \), their **statistical distance** is defined as \( \sum_{x \in X} |\chi_1(x) - \chi_2(x)| \).

### 3 Smooth expression of generators of the class group

Suppose \( \text{cl}(\mathcal{O}) \) is a cyclic group (cf. Section 3.1) of size \( N \) and with generator \( g \). We describe a procedure to find, for each positive integer \( j \), an expression \( g^{x_j} = \prod_{i=1}^t l_i^{e_i} \), with \( |e_i| \) sub-exponential with respect to \( \log N \). Here the \( l_i, i = 1, \ldots, t \) denote the first \( t \) ideals lying over Elkies primes. The idea of the algorithm is to find enough samples of the form \( g^k = \prod_{i=1}^t l_i^{e_i} \) where the \( e_i \) are chosen at random and sub-exponentially large, and then use a BKW-like algorithm to express \( g^{2^j} \) as a subexponentially short product of the samples \( g^k \).

#### 3.1 Restriction to the cyclic case

We assume the class group \( \text{cl}(\mathcal{O}) \) is cyclic. In general, the class group is not always cyclic, but heuristically it is cyclic in the vast majority of cases (97% of the time per Cohen-Lenstra [8]), and this case is easier to analyze. We conjecture that standard techniques such as [7, Appendix A] could be used to extend to the non-cyclic case.

#### 3.2 Expander graphs

It is known [12, Theorem 3.2] that isogeny graphs for (isomorphism classes of) elliptic curves with complex multiplication by some imaginary quadratic order \( \mathcal{O}_A \) where the edges are all isogenies with prime degree less than some fixed bound \( (\log |\Delta|)^B \) are in fact expander graphs. The following well-known result about expander graphs then tells us about the distribution of elliptic curves chosen from this set by taking short random walks.

**Lemma 3.1** ([12, Lemma 2.1]). Let \( \Gamma \) be a finite \( d \)-regular graph for which the non-trivial eigenvalues \( \lambda \) of the adjacency matrix are bounded by \( |\lambda| \leq c \), for some \( c < d \). Let \( S \) be any subset of the vertices of \( \Gamma \), and \( v \) any vertex in \( \Gamma \). A random walk of any length at least \( \frac{\log 2|\Gamma|/|S|^{1/2}}{\log d/c} \) starting from \( v \) will land in \( S \) with probability between \( \frac{1}{2} \frac{|S|}{|\Gamma|} \) and \( \frac{3}{2} \frac{|S|}{|\Gamma|} \).

When creating the initial state (cf. Section 4.2) we sample values \( 0 \leq k \leq |\text{cl}(\mathcal{O})| \). First, we solve \( g^{x_i} = l_i \) for each \( 1 \leq i \leq t \) in terms of our generator \( g \). If we choose random \( (e_1, \ldots, e_t) \) with \( |e_i| \in O(\log 4N) \), then we can solve \( g^k = \prod_{i=1}^t l_i^{e_i} \) by \( k = \sum_{i=1}^t a_i e_i \). The above lemma about expander graphs tells us that these \( k \)'s are chosen nearly uniformly at random from the range \( \{1, \ldots, N\} \). Notice that keeping the values of \( e_i \) bounded by this polynomial in \( \log N \) still yields subexponentially many samples for \( k \). Lastly we use a version of the BKW algorithm to compute \( g^{2^j} = \prod_{i=1}^t l_i^{e_i} \), where the exponents are subexponential in \( \log N \). In the next section we describe this algorithm in the general case.
3.3 A BKW-like algorithm

Let $n = \lceil \sqrt{\log N} \rceil$. The following two lemmas describe the iterative part of the BKW algorithm. The idea of both algorithms is to take as input a collection of uniformly chosen positive integers bounded by $N = 2^n$, and reduce the number of non-zero coefficients of their expression in base $2^n$.

**Lemma 3.2** (Upper compression). Let $k \in \{0, \ldots, n-1\}$, let $c > 0$ and let $m = (c + 1)2^n$. There exists an algorithm that takes as input $a = (a_1, \ldots, a_m) \in \{0, \ldots, 2^{n(k+1) - 1}\}$ and outputs $b = ((v_1, w_1, b_1), \ldots, (v_{m'}, w_{m'}, b_{m'}))$ where $b_i = a_{w_i} - a_{v_i} < 2^{nk}$ and $m' \geq c(2^n)$.

**Proof.** For each $i \in \{0, \ldots, 2^n - 1\}$ let $B_i$ be the set of pairs $(a_v, v)$ such that $i2^{nk} \leq a_v < (i + 1)2^{nk}$, let $c_i = \max\{a : (a, v) \in B_i\}$ and let $w_i$ be such that $(c_i, w_i) \in B_i$. Note that for any $(a_v, v) \in B_i$, $0 \leq c_i - a_v$, moreover, since $i2^{nk} \leq c_i, a_v < (i + 1)2^{nk}$, the difference is bounded by $2^{nk}$. The output of the algorithm is a vector consisting of the tuples $(v, w_i, c_i - a_v)$, with $(a_v, v) \in B_i \setminus \{(c_i, w_i)\}$, for each $i \in \{0, \ldots, 2^n - 1\}$.

**Lemma 3.3** (Lower compression). Let $k \in \{0, \ldots, n-1\}$, let $c > 0$ and let $m = (c + 1)2^n$. There exists an algorithm that takes as input a vector $a = (a_1, \ldots, a_m) \in 2^{kn} \mathbb{Z}^m$ outputs a vector $b = ((v_1, w_1, b_1), \ldots, (v_{m'}, w_{m'}, b_{m'}))$ where $2^{kn+1}$ \begin{align*} b_i = a_{w_i} - a_{v_i}, \quad \text{and } m' \geq c(2^n). \end{align*}

**Proof.** For each $i \in \{0, \ldots, 2^n - 1\}$ let $B_i$ be the set of pairs $(a_v, v)$ such that $i2^{kn} \equiv a_v \mod 2^{(k+1)n}$, let $c_i = \max\{a : (a, v) \in B_i\}$ and let $w_i$ be such that $(c_i, w_i) \in B_i$. Note that for any $(a_v, v) \in B_i$, we have $0 \leq c_i - a_v \equiv 0 \mod 2^{(k+1)n}$. The output of the algorithm is a vector consisting of the tuples $(v, w_i, c_i - a_v)$, with $(a_v, v) \in B_i \setminus \{(c_i, w_i)\}$, for each $i \in \{0, \ldots, 2^n - 1\}$.

For our purposes, we assume that the input of these algorithms is drawn from the uniform distribution [see subsection 3.2]. Suppose that one of the compression algorithms is called on an input $a$ whose entries are sampled uniformly random from $\{0, \ldots, 2^{kn-1}\}$. Notice that for any $i \in \{0, \ldots, 2^{kn-1}\}$, the expected cardinality of $B_i$ is $(c + 1)$; therefore the expected value of $c_i = \max\{a : (a, v) \in B_i\}$ is at least $\frac{(c + 1)}{2^{kn}}2^n$. This implies that the statistical distance of the distribution $c_i - a$ and uniform is $2(1 - \frac{c_i}{2^{kn+1}})$. By summing over all $i$, the expected statistical distance of the output distribution and uniform is at most $2^{n+1}(\frac{1}{2^{kn+1}})$.

Now our aim is to write $2^f$, for $f \in \{0, \ldots, n^2 - 1\}$, as a short linear combination of the given samples. The idea is to write $f = nq + r$, and call the lower compression algorithm $q$ times and the upper compression algorithm $n - q + 1$ times, to obtain samples of the form $a2^{nq}$, and find $a = 2^f$ among the samples.

**Proposition 3.4** (Iteration). Let $f \in \{0, \ldots, n^2 - 1\}$ and let $m = 2^{3n}$. There exists an algorithm $A$ that takes $a = (a_1, \ldots, a_m)$ as input, and outputs a vector $s \in \mathbb{Z}^m$ such that $\langle a, s \rangle = 2^f$, and whose expected infinity norm $||s||_\infty$ is bounded by $2^n$.

**Proof.** Let $f = nq + r$ with $0 \leq r < n$. Let $A', A''$ be the algorithms described in lemmas 3.2 and 3.3, respectively. The algorithm $A$ starts by initializing $a^{(0)} = (a_1, \ldots, a_m)$. For $i = 1, \ldots, q$, $A$ calls $A''$ on input $a^{(i-1)}$ to obtain an output $b^{(i)} = ((v_1^{(i)}, w_1^{(i)}, b_1^{(i)}), \ldots, (v_{m^{(i)}}^{(i)}, w_{m^{(i)}}^{(i)}, b_{m^{(i)}}^{(i)}))$ and sets $a^{(i)} = (b_1^{(i)}, \ldots, b_{m^{(i)}}^{(i)})$. For $j = 0, \ldots, n - q + 2$, the algorithm calls $A'$ on input $a^{(i+j)}$ to obtain an output $b^{(i+j)} = ((v_1^{(i+j)}, w_1^{(i+j)}, b_1^{(i+j)}), \ldots, (v_{m^{(i+j)}}^{(i+j)}, w_{m^{(i+j)}}^{(i+j)}, b_{m^{(i+j)}}^{(i+j)}))$ and sets $a^{(i+j)} = (b_1^{(i+j)}, \ldots, b_{m^{(i+j)}}^{(i+j)})$. By lemmas 3.2 and 3.3, the length $m^{(i+j)}$ of $b^{(i+j)}$ is $(2^{n} - n + 1)2^{n}$, and its entries are of the form $a2^{nq}$, for $a \in \{0, \ldots, 2^{n} - 1\}$. Moreover, following the discussion above, the distribution of $a$ in this set is statistically close to uniform; therefore we can find $2^f$ with high probability. Without loss of generality assume $a_0^{(n-1)} = 2^f$; then by definition we have that $2^f$ is written as a difference of two entries of $a^{(n-2)}$. Following this recursively, after $n - 1$ steps we can find $2^f$ as a linear combination of $2^n$ (possibly repeated) entries of $a$. Hence the largest coefficient of the linear combination is bounded by $2^n$.

Each of the compression steps takes $O(2^{3n})$ in time and $2^{3n}$ in space. Therefore the overall complexity is $O((n - 1)2^{3n})$ in time and space.
4 The Algorithm

4.1 Quantumly Instantiating the Action of $\text{cl}(\mathcal{O})$ in Polynomial Space

Since [4, Alg. 2] for computing the action of $\text{cl}(\mathcal{O})$ on $\mathcal{O}_p$ is not amenable to being instantiated quantumly, we present a modified algorithm here. While [4, Alg. 2] succeeds with probability 1 but has variable time, our algorithm has (tunable) fixed time but succeeds with (tunable) probability less than 1.

To begin, we give an algorithm for computing $E_B = \ell^{t_1} \ast E_A$ for prime $\ell$. We emphasize that this (classical) algorithm is designed with translation to a quantum algorithm—rather than efficiency—in mind.

Algorithm 1 succeeds if and only if there is $t' \in \{1, 2, \ldots, r\}$ such that

1. \[
\left( \frac{x_i^2 + Ax_j^2 + x_j}{p} \right) = (-1)^s; \text{ and,}
\]

2. $\ell \mid \text{ord}_{E_A}(P_r)$.

For uniformly random $x$, these conditions hold with probability $\tfrac{1}{2}$ and $\tfrac{3}{4}$, respectively, since $E(F_p) \cong \mathbb{Z}/4\mathbb{Z} \oplus \bigoplus_{i=1}^4 \mathbb{Z}/\ell_i \mathbb{Z}$. Thus the total probability that Algorithm 1 succeeds is $1 - \left( \frac{3}{4} \right)^r$. Later we shall choose a value of $k$ so that our final quantum algorithm succeeds with sufficient probability.

Next we build upon Algorithm 1 to construct an algorithm which computes $E_B = \ell^{t_e} \ast E_A$ for $e \in \mathbb{N}$. It is easy to see that Algorithm 2 succeeds with probability $\left( 1 - \left( \frac{3}{4} \right)^r \right)^e$.

**Algorithm 1** A classical algorithm for computing $\ell^{(-1)^s} \ast E_A$ for prime $\ell$, suitable for implementing on a quantum computer.

**Input:** $A \in F_p$, and $s \in \{0, 1\}$

**Output:** $B \in F_p^r$ such that $\ell^{(-1)^s} \ast E_A = E_B$, where $E_B : y^2 = x^3 + Bx^2 + x$

1: $x_1, x_2, \ldots, x_r \leftarrow U(F_p), \ c \leftarrow 0$

2: for $i$ from 1 to $r$ by 1 do

3: \[y_i \leftarrow \sqrt{x_i^2 + Ax_j^2 + x_j} \] \text{\hspace{1cm} ▷ In the extension field $F_p^r$.}

4: \[P_i \leftarrow (x_i, y_i), \ Q_i \leftarrow \left[ \frac{p+1}{r} \right] P_i \]

5: if $(x_i^2 + Ax_j^2 + x_j) = (-1)^s$ and $Q_i \neq \infty$ and $c = 0$ then

6: Compute $E_B$, where $\phi : E_A \to E_B : y^2 = x^3 + Bx^2 + x$ is an isogeny with ker $\phi = \langle Q_i \rangle$

7: end if

8: if $(x_i^2 + Ax_j^2 + x_j) = (-1)^s$ and $Q_i \neq \infty$ then

9: \[c \leftarrow c + 1\]

10: end if

11: end for

**Algorithm 2** A classical algorithm for computing $\ell^{(-1)^s} \ast E_A$ for prime $\ell$, suitable for implementing on a quantum computer.

**Input:** $A \in F_p$, $s \in \{0, 1\}$, and $e \in \mathbb{N}$

**Output:** $B \in F_p^r$ s.t. $\ell^{(-1)^s} \ast E_A = E_B$, where $E_B : y^2 = x^3 + Bx^2 + x$

1: $x_1, x_2, \ldots, x_r \leftarrow U(F_p), \ e' \leftarrow e, \ B \leftarrow A$

2: for $j$ from 1 to $e$ by 1 do

3: if $e' > 0$ then

4: \[B \leftarrow C, \text{ where } \ell^{(-1)^s} \ast E_B = E_C : y^2 = x^3 + Cx^2 + x, \text{ computed using random values } x_1, x_2, \ldots, x_r\]

5: end if

6: \[e' \leftarrow e' - 1\]

7: end for
Algorithm 3 An algorithm for computing \( t_1^{e_1} t_2^{e_2} \cdots t_t^{e_t} A \) for primes \( \ell_1, \ell_2, \ldots, \ell_t \), suitable for implementing on a quantum computer.

**Input:** \( A \in F_p, s \in \{0, 1\}^t, \) and \( e \in \mathbb{N}^t \)

**Output:** \( B \in F_p \) such that \( t_1^{e_1} t_2^{e_2} \cdots t_t^{e_t} A = B, \) where \( B : y^2 = x^3 + Bx^2 + x \)

1. \( x_1, x_2, \ldots, x_r \leftarrow U(F_p) \)
2. for \( k \) from 1 to \( t \) by 1 do
3. \( B \leftarrow C, \) where \( t_k^{e_k} A = E_B : y^2 = x^3 + Cx^2 + x, \) computed using random values \( x_1, x_2, \ldots, x_r. \)
4. end for

Finally, Algorithm 3 computes \( t_1^{e_1} t_2^{e_2} \cdots t_t^{e_t} A. \)

Then \( P[\text{Algorithm 2 succeeds}] = \prod_{k=1}^r \left( 1 - \left( \frac{Q_{i+1}}{2^{Q_i}} \right)^{e_k} \right) \geq \left( 1 - \left( \frac{2}{1} \right) \right)^{\|e\|_1}. \) From here we briefly describe how to instantiate this quantumly. First we describe the quantum instantiation of Algorithm 1. For brevity of notation we consider an input \( |s\rangle|E_A\rangle|0\rangle \) which is not in superposition, but of course the algorithm extends linearly to superpositions. Before the quantum part of the algorithm begins, we sample \( x_1, x_2, \ldots x_r \leftarrow U(F_p) \) classically and include them as part of the initial state. We will use them in the quantum instantiation of all three algorithms.

(i) In the notation of Algorithm 1, write \( (Q_i)_{i=1}^t \) to a new register to obtain
\[
|s\rangle|E_A\rangle|x_1, x_2, \ldots, x_r\rangle|Q_1, Q_2, \ldots, Q_t\rangle|0\rangle
\]

(ii) In the notation of Algorithm 1, define \( w_i = 1 \) if \( \left( \frac{x_i^3 + Ax^2 + x}{p} \right) = (-1)^s \) and \( Q_i \neq \infty, \) and \( w_i = 0 \) otherwise. Write \( w_1, w_2, \ldots, w_r \) to a new register to obtain
\[
|s\rangle|E_A\rangle|x_1, x_2, \ldots, x_r\rangle|Q_1, Q_2, \ldots, Q_t\rangle|w_1, w_2, \ldots, w_r\rangle|0\rangle.
\]

This can be done by writing the results of each of the Boolean functions \( \left( \frac{x^3 + Ax^2 + x}{p} \right) = (-1)^s \) and \( [Q_i \neq \infty] \) to new registers, applying a Toffoli gate from these registers onto another new register, and then uncomputing the results of the two Boolean functions.

(iii) Set aside a new 0-initialized register to contain \( c. \) For \( i = 1, 2, \ldots, r \) apply Vélu’s formulas [21] conditioned on \( w_i = 1 \) and \( c = 0, \) and increment \( c \) conditioned on \( w_i = 1. \) Essentially, we look down the list of points until we find \( x_r \) which is “appropriate” (has \( w_r = 1 \)) and, when we find the first appropriate point, we set a flag which indicates that we have computed \( l \times E_A, \) and so we should not compute more. The resultant state is
\[
|s\rangle|E_A\rangle|x_1, x_2, \ldots, x_r\rangle|Q_1, Q_2, \ldots, Q_r\rangle|w_1, w_2, \ldots, w_r\rangle|\hat{c}\rangle|l \times E_A\rangle|0\rangle.
\]

where \( \hat{c} \) counts the number of \( w_i \) which are equal to 1.

(iv) Uncompute \( \hat{c} \) by subtracting 1 from it conditioned on \( w_i = 1, \) for \( i = r, r - 1, \ldots, 1. \) Then uncompute \( (Q_i)_{i=1}^t \) and \( (w_i)_{i=1}^t \) by reversing the circuit of step (ii). Rearranging the registers, the final state is
\[
|s\rangle|E_A\rangle|l \times E_A\rangle|x_1, x_2, \ldots, x_r\rangle|0\rangle,
\]
as required.

For fixed \( \ell, \) call the algorithm above \( \Omega^1_\ell. \) We shall use it as a subroutine in the quantum instantiation of Algorithm 2. For input \( |s\rangle|e\rangle|E_A\rangle|x_1, x_2, \ldots, x_r\rangle|0\rangle: \)

(i) Conditioned on register 2 being positive, apply \( \Omega^1_\ell \) to registers 1, 3, and 4 targeting a new register. Then decrement register 2, yielding
\[
|s\rangle|e - 1\rangle|E_A\rangle|x_1, x_2, \ldots, x_r\rangle|l \times E_A\rangle|0\rangle
\]

(ii) Swap registers 2 and 5, and conditioned on register 2 being positive, apply \( \Omega^1_\ell \) again to registers 1, 2, 3, and 4 targeting a new register. Decrement register 2 to obtain
\[
|s\rangle|e - 2\rangle|l \times E_A\rangle|x_1, x_2, \ldots, x_r\rangle|E_A\rangle|l \times E_A\rangle|0\rangle.
\]
Given a state \(|s⟩ (e - z)| (l^{(t-1)(e-1)} E_A) |x_1, x_2, \ldots, x_r⟩ (l^{(t-1)e} E_A) |0⟩\), swap registers 3 and 6, and apply the Pauli X to register 1. This yields \(|1 - s⟩ (e - z)| (l^{(t-1)(e-1)} E_A) |x_1, x_2, \ldots, x_r⟩ (l^{(t-1)e} E_A) |0⟩\). Apply \(Q_{l}^{(t)}\) to registers 1, 2, 3, and 4 targeting register 6 conditioned on register 2 being positive. Notice that the output of \(Q_{l}^{(t)}\) in this case is \((l^{(t-1)(e-1)} E_A)\) since we are applying \((l^{(t-1)e})\) in this case. This effectively erases the contents of register 6. We can then apply Pauli X to register 1 again, and, conditioned on register 2 being positive, we apply \(Q_{l}^{(t)}\) and decrement register 2 to obtain \(|s⟩ (e - (z + 1)) (l^{(t-1)(e-1)} E_A) |0⟩\).

(iv) Repeat step (iii) \(L - 2\) times, where \(L \geq e\). The result is

\[|s⟩ (e - L) (l^{(t-1)(e-1)} E_A) |x_1, x_2, \ldots, x_r⟩ (l^{(t-1)e} E_A) |0⟩\]

Copy register 6 onto a new register to obtain

\[|s⟩ (e - L) (l^{(t-1)(e-1)} E_A) |x_1, x_2, \ldots, x_r⟩ (l^{(t-1)e} E_A) |0⟩\].

From here, we can simply reverse the iterations of step (iii) and steps (ii) and (i), erasing the ancillary registers. Rearranging registers yields

\[|s⟩ |E_A⟩ (l^{(t-1)e} E_A) |x_1, x_2, \ldots, x_r⟩ |0⟩\], as required.

For fixed \(l\), call this algorithm \(Q_{l}^{(L)}\). To evaluate \((l^{(t-1)e_1} E_A l^{(t-1)e_2} E_A \cdots l^{(t-1)e_l} E_A)\), it suffices to apply each \(Q_{l}^{(L)}\) in turn to the appropriate registers of

\[|s_1, s_2, \ldots, s_t⟩ |e_1, e_2, \ldots, e_l⟩ |E_A⟩ |x_1, x_2, \ldots, x_r⟩ |0⟩\]

where \(L \geq e\). It is easy to see that this computes the correct value.

**Remark 4.1.** When we want to compute the action in superposition, we need to apply \(Q_{l}^{(L)}\) for \(L\) greater than all the \(e\) values supported in the superposition. For unknown states, this is not possible, but for our purposes it suffices to be able to compute the action for known superpositions; we can then choose \(L\) appropriately.

### 4.2 Constructing the States

In this subsection we show how to use the algorithms previously described to construct the states required to apply Kuperberg’s algorithm.

Given curves \(E = E_A : y^2 = x^3 + Ax^2 + x\) and \(E' = E_B : y^2 = x^3 + Bx^2 + x\) where \(E_B = a \times E_A\), and \(a = g^a\), Kuperberg’s algorithm uses states of the form \(|ψ_k⟩ := \frac{1}{\sqrt{2}} \left( |0⟩ + \exp \left( \frac{2\pi iαk}{N} \right) |1⟩ \right)\) for \(k\) sampled uniformly at random from \(\{0, 1, \ldots, N - 1\}\).

Using the method described in Section 3 we can construct a table \((2^t, \mathbf{v}(j)) \in \mathbb{N} \times \mathbb{Z}^t, \|\mathbf{v}(j)\|_{∞} = 2O(n)\) and \([g^2]^t = [v^0_1, v^0_2, \ldots, v^0_t] = [v^0_{j1}, v^0_{j2}, \ldots, v^0_{jt}]\) for \(1 ≤ j ≤ N\). From this table, we construct the following quantum circuit, which converts from “cyclic notation” \(g^m\) to “prime decomposition notation” \(v^0_{j1} v^0_{j2} \cdots v^0_{jt}\):

```
| m_0 ⟩   | m_1 ⟩   | ⋯   | m_n ⟩
| ⋮       | ⋮       | ⋮   | ⋮       |
| 0       | +v₀   | +v₁   | +v_n |
| 0       |       |       |       |
```

```circuit
C
```

where \(m_i\) represents the cyclic notation for the input value, \(v_j\) represents the prime decomposition notation, and \(C\) is the quantum circuit that performs the conversion.
Remark 4.2. In step (iv) we evaluate the class group action on a uniform superposition over \( \text{cl}(0) \times \mathbb{Z}/2\mathbb{Z} \) (with the second coordinate determining to which curve we apply the element of \( \text{cl}(0) \)) with fixed randomness for each such input, using the prime decomposition presentation of the group elements. We find that the probability of evaluating the function correctly over the entire superposition is\( N \max_{h \in \text{cl}(0)} \|v(h)\|_1 \) in particular, the success probability is negligibly different from 1 using only a polynomial number of random points \( \{x_1\}_i \).
4.3 Using the States to Find the Hidden Shift

Now that we have a method to obtain states of the form $|\psi_k\rangle$ for uniformly random $k$, we can apply Regev’s sieve [16]1 to extract the states $|\psi_1\rangle, |\psi_2\rangle, \ldots, |\psi_{2^m-1}\rangle$ where $m = \lceil \log_2 N \rceil$. Unlike Kuperberg’s original algorithm [13], Regev’s sieve requires only polynomial quantum storage.

From here we proceed using [13, Remark 5.2]; we note that $\mathcal{O}_{k=0}^{m-1} |\psi_{2^k}\rangle = 2^{-\frac{3}{2}} \sum_{y=0}^{2^m-1} \omega^{ay} |y\rangle$, and $F_N|a\rangle = N^{-\frac{1}{2}} \sum_{y=0}^{N-1} \omega^{ay} |y\rangle$ where $F_N$ is the quantum Fourier transform. Since these states have inner product $\frac{x^T}{N} = \Omega(1)$ and this inner product is preserved by the inverse Fourier transform, it follows that measuring $F_N^{(a)} \otimes_{j=0}^{k-1} |\psi_j\rangle$ in the computational basis will yield $a$ with probability $\Omega(1)$.

4.4 Time and Space Analysis

We briefly explain the time and space analysis of the algorithms. The classical portions of the algorithm are:

(i) Generating a subexponential number $m$ of samples $\sum_{i=1}^m a_i e_i = k$, for small random $e_i$ and known $a_i = \log_2 \ell_i$.

(ii) Using a BKW-like algorithm to find an expression $\sum_{j=1}^m s_j k_j = 2^t$, with $s_j$ subexponential and $k_j$ from the given samples above. Its time and space complexity is $O(2^{3\lceil \log N \rceil})$.

As for the quantum portion, we have:

(i) For a given $t$, algorithm 1 runs in time $O(r \ell \log p)$ and uses quantum space $O(r \log p) = O(\log^{2.5} N \log p)$.

(ii) For Algorithm 2, we repeat Algorithm 1 $2O(\sqrt{\log N})$ times. So this algorithm runs in time

$$O(r \ell_{\text{max}} \log p) 2^{O(\sqrt{\log N})} = 2^{O(\sqrt{\log N})}$$

(for $r, \ell_{\text{max}}$, $\log p = 2^{O(\sqrt{\log N})}$ and space $O(\log^{2.5} N \log p)$).

(iii) For Algorithm 3, apply Algorithm 2 $t$ times. This requires time $2^{O(\sqrt{\log N})}$ (for $t = 2^{O(\sqrt{\log N})}$) and space $O(\log^{2.5} N \log p)$.

(iv) Each sample in Regev’s algorithm [16] invokes Algorithm 3 once, and so all our calls to Algorithm 3 take total time $2^{O(\sqrt{\log N} \log \log N)}$. The space complexity is $O(\log^3 N \log p \sqrt{\log N})$.

Acknowledgement: This research was facilitated by the Banff International Research Station (BIRS) Workshop on “An Algebraic Approach to Multilinear Maps for Cryptography,” May 6 to 11, 2018. We thank Daniel J. Bernstein, Jean-François Biasse, Xavier Bonnetain, Steven Galbraith, Tanja Lange, Barak Shani and Mark Zhandry for helpful discussions, and the referees for useful feedback.

This research was undertaken thanks in part to funding from the Canada First Research Excellence Fund, CryptoWorks21, Public Works and Government Services Canada, and the Royal Bank of Canada.

References

[1] Jean-François Biasse, Annamaria Iezzi and Michael J. Jacobson Jr., A note on the security of CSIDH, 2018.

[2] Avrim Blum, Adam Kalai and Hal Wasserman, Noise-tolerant Learning, the Parity Problem, and the Statistical Query Model, J. ACM 50 (2003), 506–519.

[3] Xavier Bonnetain and André Schrottenloher, Quantum Security Analysis of CSIDH and Ordinary Isogeny-based Schemes, Cryptology ePrint Archive, Report 2018/537, 2018, https://eprint.iacr.org/2018/537.

[4] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny and Joost Renes, CSIDH: An Efficient Post-Quantum Commutative Group Action, Cryptology ePrint Archive, Report 2018/383, 2018, https://eprint.iacr.org/2018/383.

---

1 Regev only explicitly deals with the case where the underlying cyclic group is of order $2^{k^2+1}$ for integer $k$, but the method can be extended (as in [7, Appendix A]) to arbitrary cyclic groups.
[5] Denis X. Charles, Kristin E. Lauter and Eyal Z. Goren, Cryptographic Hash Functions from Expander Graphs, *Journal of Cryptology* 22 (2009), 93–113.

[6] Andrew Childs, David Jao and Vladimir Soukharev, Constructing elliptic curve isogenies in quantum subexponential time, eprint arXiv:quant-ph/0406151, 2010, https://arxiv.org/abs/1012.4019v3.

[7] Andrew Childs, David Jao and Vladimir Soukharev, Constructing elliptic curve isogenies in quantum subexponential time, *J. Math. Cryptol* 8 (2014), 1–29.

[8] Henri Cohen and Hendrik W. Lenstra Jr., Heuristics on class groups of number fields, *Number theory, Noordwijkerhout 1983 (Noordwijkerhout, 1983)*, Lecture Notes in Math. 1068 (1984), 33–62.

[9] Jean-Marc Couveignes, *Hard Homogeneous Spaces*, Cryptology ePrint Archive, Report 2006/291, 2006, https://eprint.iacr.org/2006/291.

[10] Luca De Feo, Jean Kieffer and Benjamin Smith, Towards practical key exchange from ordinary isogeny graphs, Cryptology ePrint Archive, Report 2018/485, 2018, https://eprint.iacr.org/2018/485.

[11] Steven D. Galbraith and Frederik Vercauteren, Computational problems in supersingular elliptic curve isogenies, Cryptology ePrint Archive, Report 2017/774, 2017, https://eprint.iacr.org/2017/774.

[12] David Jao, Stephen D. Miller and Ramarathnam Venkatesan, Expander graphs based on GRH with an application to elliptic curve cryptography, *J. Number Theory* 129 (2009), 1491–1504.

[13] Greg Kuperberg, A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem, *SIAM Journal on Computing* 35 (2005), 170–188.

[14] Greg Kuperberg, Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem, in: 8th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2013) (Simone Severini and Fernando Brandao, eds.), Leibniz International Proceedings in Informatics (LIPIcs) 22, pp. 20–34, Schloss Dagstuhl, Dagstuhl, Germany, 2013.

[15] Arjen K. Lenstra, Hendrik W. Lenstra and László Lovász, Factoring polynomials with rational coefficients, *Mathematische Annalen* 261 (1982), 515–534 (English).

[16] Oded Regev, A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space, *eprint arXiv:quant-ph/0406151* (2004).

[17] Oded Regev, *A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space*, 2004.

[18] Alexander Rostovtsev and Anton Stolbunov, Public-Key Cryptosystem Based on Isogenies, *IACR Cryptology ePrint Archive 2006* (2006), 145.

[19] Claus-Peter Schnorr and M. Euchner, Lattice basis reduction: Improved practical algorithms and solving subset sum problems, *Mathematical Programming* 66 (1994), 181–199.

[20] Anton Stolbunov, Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves, *Adv. Math. Commun.* 4 (2010), 215–235.

[21] Jacques Vélu, Isogénies entre courbes elliptiques, *C. R. Acad. Sci. Paris Sér. A-B* 273 (1971), 238–241.