He Gives C-Sieves on the CSIDH

Chris Peikert*

June 27, 2019

Abstract

Recently, Castryck, Lange, Martindale, Panny, and Renes proposed *CSIDH* (pronounced "sea-side") as a candidate post-quantum "commutative group action." It has attracted much attention and interest, in part because it enables noninteractive Diffie–Hellman-like key exchange with quite small communication. Subsequently, CSIDH has also been used as a foundation for digital signatures.

In 2003–04, Kuperberg and then Regev gave asymptotically subexponential quantum algorithms for "hidden shift" problems, which can be used to recover the CSIDH secret key from a public key. In late 2011, Kuperberg gave a follow-up quantum algorithm called the *collimation sieve* ("c-sieve" for short), which improves the prior ones, in particular by using exponentially less quantum memory and offering more parameter tradeoffs. While recent works have analyzed the concrete cost of the original algorithms (and variants) against CSIDH, nothing of this nature was previously available for the c-sieve.

This work fills that gap. Specifically, we generalize Kuperberg's collimation sieve to work for arbitrary finite cyclic groups, provide some practical efficiency improvements, give a classical (i.e., non-quantum) simulator, run experiments for a wide range of parameters up to and including the actual CSIDH-512 group order, and concretely quantify the complexity of the c-sieve against CSIDH.

Our main conclusion is that the proposed CSIDH-512 parameters provide relatively little quantum security beyond what is given by the cost of quantumly evaluating the CSIDH group action itself (on a uniform superposition). The cost of key recovery is, for example, only about 2^{16} quantum evaluations using 2^{40} bits of quantumly accessible *classical* memory (plus insignificant other resources). This improves upon a prior estimate of $2^{32.5}$ evaluations and 2^{31} qubits of *quantum* memory, for a variant of Kuperberg's original sieve. Under the plausible assumption that quantum evaluation does not cost much more than is indicated by a recent "best case" analysis, CSIDH-512 can therefore be broken using significantly less than 2^{64} quantum T-gates (plus relatively small other resources). This falsifies a standard interpretation of the claimed 64 bits of quantum security, and it strongly invalidates the claimed NIST security level 1 when accounting for the MAXDEPTH restriction.

*Computer Science and Engineering, University of Michigan. Email: `cpeikert@umich.edu`. This material is based upon work supported by the Patrick C. Fischer Development Chair and the National Science Foundation under Award CNS-1606362. The views expressed are those of the author and do not necessarily reflect the official policy or position of the National Science Foundation.

1 Introduction

In 1994, Shor [Sho94] upended cryptography by giving polynomial-time *quantum* algorithms for the integer factorization and discrete logarithm problems, which can be used (on sufficiently large-scale quantum computers) to break all widely deployed public-key cryptography. With the steady progress in engineering quantum computers, there is an increasing need for viable *post-quantum* cryptosystems, i.e., ones which can be run on today's classical computers but resist attacks by future quantum ones. Indeed, the US National Institute of Standards and Technology (NIST) has begun a post-quantum standardization effort [NIS], and recently selected the second-round candidates.

1.1 Isogeny-Based Cryptography

One prominent class of candidate post-quantum cryptosystems uses *isogenies* between elliptic curves over a common finite field. Isogeny-based cryptography began with the proposal of Couveignes in 1997, though it was not widely distributed until 2006 [Cou06]. The approach was independently rediscovered by Stolbunov (in his 2004 Master's thesis [Sto04]) and by Rostovtsev and Stolbunov [RS06] in 2006. The central object in these proposals is a (free and transitive) *group action* $\star\colon G \times Z \to Z$ of a finite *commutative* group G on a set Z. Group actions naturally generalize exponentiation in (finite) cyclic multiplicative groups C: we take $G = \mathbb{Z}_q^*$ to be the multiplicative group of integers modulo the order $q = |C|$ and Z to be the set of generators of C, and define $a \star z = z^a$.

The Couveignes–Rostovtsev–Stolbunov (hereafter CRS) proposal very naturally generalizes Diffie–Hellman [DH76] noninteractive key exchange to use a commutative group action: some $z \in Z$ is fixed for use by all parties; Alice chooses a secret $a \in G$ and publishes $p_A = a \star z$; Bob likewise chooses a secret $b \in G$ and publishes $p_B = b \star z$; then each of them can compute their shared key $(ab) \star z = a \star p_B = b \star p_A$. (Note the essential use of commutativity in the second equation, where $b \star (a \star z) = (ba) \star z = (ab) \star z$.)

Security. Of course, for the CRS system to have any hope of being secure, we need the analogue of the discrete logarithm problem for the group action to be hard, i.e., it must be infeasible to recover a (or some functional equivalent) from $p_A = a \star z$. In 2010, Childs, Jao, and Soukharev [CJS10] observed that, assuming a suitable algorithm for the group action, this problem reduces to the (injective) abelian hidden-shift problem on the group G. It happens that Kuperberg [Kup03] in 2003 and then Regev [Reg04] in 2004 had already given asymptotically subexponential quantum "sieve" algorithms for this problem. More specifically, Kuperberg's algorithm uses $\exp(O(\sqrt{n}))$ quantum time and space, whereas Regev's uses slightly larger $\exp(O(\sqrt{n \log n}))$ quantum time but only $\mathrm{poly}(n)$ quantum space, where $n = \log N$ is the bit length of the group order $N = |G|$. While these attacks do not render CRS-type systems insecure, one must consider their concrete complexity when setting parameters to obtain a desired level of security.

We mention that these subexponential attacks against CRS motivated Jao and De Feo [JD11] to give a different approach to isogeny-based cryptography using *supersingular* curves, whose full endomorphism rings are non-commutative, which thwarts the Kuperberg-type attacks. The Jao–De Feo scheme, now known as Supersingular Isogeny Diffie–Helmman (SIDH), is also not based on a group action, and is inherently interactive. Most research on isogeny-based cryptography has focused on SIDH and closely related ideas.

CSIDH. The noninteractive nature and simplicity of the CRS approach are particularly attractive features, which motivated Castryck, Lange, Martindale, Panny, and Renes [CLM$^+$18] to revisit the method recently. They proposed "Commutative SIDH," abbreviated CSIDH and pronounced "sea-side." Like SIDH, it relies on supersingular curves, but it uses a *commutative subring* of the full endomorphism ring, which naturally leads to a commutative group action. This design choice and other clever optimizations yield an impressive efficiency profile: for the CSIDH-512 parameters that were claimed in [CLM$^+$18] to have 64 bits of quantum security and meet NIST security level 1, a full key exchange takes only about 80 milliseconds (improving upon several minutes for prior CRS prototypes), with key sizes of only 64 bytes (compared to hundreds of bytes for SIDH and derivatives).

In summary, the designers of CSIDH describe it as a primitive "that can serve as a drop-in replacement for the (EC)DH key-exchange protocol while maintaining security against quantum computers." As such, it has attracted a good deal of attention and interest. (For example, it received the 2019 Dutch Cybersecurity Research Paper Award.) In addition, a series of works [Sto11, DG19, BKV19] used CSIDH to develop digital signature schemes having relatively small sizes and reasonable running times. E.g., for the same claimed security levels as above, the CSI-FiSh signature scheme [BKV19] can have a combined public key and signature size of 1468 bytes, which is better than all proposals to the NIST post-quantum cryptography effort.

1.2 Attacks on the CSIDH

As mentioned above, when setting parameters for CSIDH and arriving at security claims, one must take into account known attacks. The main quantum approach is given by Kuperberg's abelian hidden-shift algorithm [Kup03] and descendants, where the hidden "shift" corresponds to the secret "discrete log" $a \in G$ for a given public key $p_A = a \star z \in Z$. Algorithms of this type have two main components:

1. a quantum *oracle* that, whenever queried, outputs a certain kind of random "labeled" quantum state, in part by evaluating the group action on a uniform superposition over the group;
2. a *sieving* procedure that combines labeled states in some way to generate "more favorable" ones.

By processing many fresh labeled states from the oracle, the sieve eventually creates some "highly favorable" states, which are then measured to reveal useful information about the hidden shift (i.e., the secret key).

The overall complexity of the attack is therefore mainly determined by the complexities of the quantum oracle and the sieve, where the latter includes the oracle query complexity. These can be analyzed independently, and for each there is a line of work with a focus on CRS/CSIDH.

The oracle. To produce a labeled state, the oracle mainly needs to prepare a uniform superposition over the group G, and apply the group action to a superposition of the "base" $z \in Z$ and the public key $a \star z$. (It then does a certain measurement, takes a Fourier transform, and measures again to get a label.) In the context of isogenies, evaluating the group action on the superposition is presently the bottleneck, by a large amount.

The original work of Childs, Jao, and Soukharev [CJS10] implemented the oracle in $\exp(\tilde{O}(n^{1/2}))$ quantum time (assuming GRH) and polynomial space. Biasse, Iezzi, and Jacobson [BIJJ18] improved this to an oracle that (under different heuristics) runs in $\exp(\tilde{O}(n^{1/3}))$ quantum time and polynomial space, though they did not analyze the factors hidden by the $\tilde{O}$ notation.

More recently, Bernstein, Lange, Martindale, and Panny [BLMP19] analyzed the *concrete* cost of quantumly evaluating the CSIDH group action. For the CSIDH-512 parameters, they arrived at an estimate of less than 2^{40} nonlinear bit operations (which translates to between 2^{40} and 2^{44} quantum T-gates), with a failure probability below 2^{-32}, to evaluate the group action on a non-uniform "best conceivable" (for the attacker) distribution of group elements, namely, the one used in CSIDH key generation. Very recent work by Beullens, Kleinjung, and Vercauteren [BKV19] suggests that the cost for a *uniform* superposition may be quite close to that of the "best conceivable" case; see Section 1.4 for further discussion.

The sieve. Kuperberg's original algorithm [Kup03] has $\exp(O(\sqrt{n}))$ complexity in time, queries, *and* quantum space. More specifically, he rigorously proved a query bound of $O(2^{3\sqrt{n}})$, and a better time and query bound of $\tilde{O}(3^{\sqrt{2\log_3 N}})$ when $N = r^n$ for some small radix r (though this is very unlikely to be the case for CSIDH). As already mentioned, Regev reduced the quantum space to only polynomial in n, but at the cost of increasing the time and query complexity to $\exp(O(\sqrt{n \log n}))$; to our knowledge, precise hidden factors have not been worked out for this approach.

Bonnetain and Schrottenloher [BS18] provided a variant of Kuperberg's sieve for arbitrary cyclic groups, and gave more precise estimates of its query and quantum-space complexity. Specifically, using simulations up to $n = 100$ they estimate that $2^{1.8\sqrt{n}+2.3}$ queries and nearly the same number of qubits of memory are needed. For the CSIDH-512 parameters, this translates to $2^{32.5}$ queries and 2^{31} qubits.

Notably, in late 2011 Kuperberg gave a follow-up algorithm [Kup11], called the *collimation sieve* (or "c-sieve" for short), which subsumes his original one and Regev's variant.[1] Asymptotically, it still uses $\exp(O(\sqrt{n}))$ quantum time and classical space, but only *linear* $O(n)$ quantum space (in addition to the oracle's). Moreover, it provides other options and tradeoffs, most notably between classical and quantum time when using *quantumly accessible classical* memory (QRACM, also known as QROM), i.e., classical memory that is readable (but not writeable) in superposition. As argued in [BHT98, Kup11], QRACM is plausibly much cheaper than fully quantum memory, because it does not need to be preserved in superposition. In particular, Kuperberg describes [Kup11, Proposition 2.2] how QRACM can be simulated using ordinary classical memory, at the cost of logarithmic quantum memory and quasilinear quantum time in the number of data cells; see [BGB$^+$18, Section III.C] for a realization of this idea which has modest concrete cost.

Although Kuperberg's collimation sieve dates to about six years before the CSIDH proposal, and has been briefly cited in some of the prior literature on CSIDH, an analysis for concrete parameters was not previously available.[2] That is the question we address in this work.

1.3 Contributions

We analyze the concrete complexity of Kuperberg's collimation sieve [Kup11], with a focus on CSIDH and its proposed parameterizations.[3] Our study mainly treats the quantum oracle as a "black box," and focuses on the precise number of queries and amount of quantumly accessible classical memory (QRACM) the sieve uses. However, following a suggestion by Schanck [Sch19], we also give a rough analysis of how these quantities translate to the quantum complexity of the full attack.

More specifically, we generalize the c-sieve to work for cyclic groups of arbitrary finite order (from power-of-two or other smooth orders, which CSIDH groups typically do not have), provide some practical improvements that maintain better control of the memory and time complexities, give a classical simulator and run experiments on a wide range of parameters—including the actual CSIDH-512 group order of $N \approx 2^{257.1}$—and concretely quantify the complexity of the c-sieve against proposed CSIDH parameters. As far as we know, ours is the first work to simulate *any* kind of quantum sieve algorithm for groups as large as the actual CSIDH-512 group, or even groups having orders much larger than $N = 2^{100}$.

Our main conclusion is that the CSIDH-512 parameters provide relatively little quantum security beyond what is given by the cost of the quantum oracle: the secret key can be recovered from the public key with, for example, only about 2^{16} oracle queries and 2^{40} bits of QRACM, or about $2^{19.3}$ queries and 2^{32} bits of

---

[1] More recently, Kuperberg has given talks highlighting the virtues of the algorithm and its relevance to isogenies.

[2] Shortly after the announcement of this work, Bonnetain and Schrottenloher posted an updated version of [BS18], which had been under private review and which does contain such an analysis. See below for a comparison.

[3] Our work has no implications for SIDH [JD11] or the NIST submission SIKE, which do not use commutative group actions.

QRACM, plus insignificant other resources. This improves upon a prior estimate [BS18] of $2^{32.5}$ queries and 2^{31} qubits of *quantum* memory, for a variant of Kuperberg's original sieve. Moreover, we find that for the parameters of interest, the cost of implementing the QRACM is dwarfed by that of the oracle queries (under current estimates for the latter). See Section 5 for the full details.

It follows that, under the plausible assumption that implementing the oracle does not cost much more than the "best conceivable case" estimate of [BLMP19], CSIDH-512 can be broken using significantly less than 2^{64} quantum T-gates, plus relatively small other resources. This falsifies a standard interpretation of the claimed 64 bits of quantum security, and it strongly invalidates the claimed NIST security level 1 when accounting for the MAXDEPTH restriction.[4] Similarly, CSIDH-1024 and -1792, which were respectively conjectured to have at least 96 and 128 bits of quantum security, can be broken with, e.g., about 2^{27} and 2^{41} queries to an appropriate oracle and the same amount of QRACM (plus cryptanalytically plausible other resources). We again emphasize that the c-sieve offers a flexible tradeoff among queries, QRACM, and classical time, so these example query counts can be reduced somewhat by increasing these other resources.

Comparison with [BS18]. As mentioned above, shortly after the announcement of this work, Bonnetain and Schrottenloher posted an update [BS18] to their earlier analysis of Kuperberg's first sieve algorithm, which also analyzes variants of the collimation sieve. They arrive at similar conclusions, but their analysis is largely complementary to ours, in the following ways. They give a theoretical analysis that ignores some polynomial terms, whereas ours is fully concrete and supported by experiments (which reveal some unexpected phenomena that affect the polynomial factors). They also consider large collimation "arity" (see below) with correspondingly small fully quantum memory and large classical work and memory, whereas we mainly limit our attention to the binary case $r = 2$ with correspondingly larger QRACM and small classical work. Finally, we include optimizations that are not considered in [BS18], like the extraction of many secret-key bits from each run of the sieve. It seems likely that a combination of ideas from these works would yield additional points on the attack spectrum and somewhat improved bounds.

1.4 Further Research

The main question that remains to be addressed is the concrete complexity of the quantum oracle, i.e., evaluation of the CSIDH group action for a uniform superposition over the group. The results of [BS18] and even moreso [BKV19] suggest that for CSIDH-512, the cost may be close to the roughly 2^{40} nonlinear bit operations estimate [BLMP19] for the "best conceivable case"—perhaps even within a factor of two or less. This is because [BKV19] gives a fast method for mapping a uniformly random group element to a short exponent vector, which has a very similar norm distribution to the one analyzed in [BLMP19]. (The norm's expectation is only about 10% larger, and its variance is actually somewhat smaller.) Also, because the sieve requires so few oracle queries (e.g., 2^{16} for CSIDH-512), some immediate improvement may be obtainable simply by increasing the oracle's error probability, from the 2^{-32} considered in [BLMP19]. A related question is whether it is possible to accelerate the oracle computations by amortization.

Our study is primarily focused on collimation *arity* $r = 2$, which corresponds to a sieve that produces a binary recursion tree. Using an arity $r > 2$ can reduce the number of queries and/or the needed amount of QRACM, at the cost of more classical time. In a bit more detail, the main collimation subroutine that for $r = 2$ takes quasilinear $\tilde{O}(L)$ classical time (in the amount L of QRACM) takes $\tilde{O}(L^{r-1})$ classical time

[4] We note that [CLM+18] does not explicitly state the metric by which its "bits of quantum security" claims should be evaluated, but its Section 7.3 and Table 1 use "number of qubit operations," which are frequently separated into Clifford gates and much more expensive T-gates (see, e.g., [BLMP19]). Other complexity metrics, like "depth times width," can be considered as well; see, e.g., [JS19] for a treatment of this issue.

in general (or even less time with more memory, using Schroeppel–Shamir [SS79]), but reduces the depth of the recursion tree by about an $r-1$ factor, which can significantly reduce the number of oracle queries. Our experiments demonstrate that the classical work for $r=2$ is cryptanalytically small (on the order of several core-days), so larger arities seem worth investigating, especially if the quantum oracle remains the main bottleneck.

A final interesting question is how many bits of a CSIDH secret are required to break the scheme. Our complexity estimates are for running the c-sieve several times to recover almost all of the secret bits (the remainder can be obtained by brute force). However, if partial information about the secret suffices to break the scheme through other means, then the number of sieve invocations and corresponding query complexity would be reduced.

1.5 Paper Organization

In Section 3 we describe and analyze our generalization of Kuperberg's collimation sieve to arbitrary cyclic groups. In Section 4 we describe our classical simulator for the collimation sieve, and report on our experiments with it. In Section 5 we draw conclusions about the quantum security of various CSIDH parameters.

Acknowledgments. We thank the organizers of the Oxford Post-Quantum Cryptography Workshop, at which we received our first detailed exposure to CSIDH; Michael Cafarella and J. Alex Halderman for the use of their servers to conduct experiments; Oded Regev and Greg Kuperberg for clarifying comments; Dan Bernstein for pointing out Kuperberg's recent talks and for comments on [BLMP19]; Léo Ducas for additional references, and John Schanck for additional references and discussions about the full quantum complexity analysis.

2 Preliminaries

We let $\mathbb{N} = \{0, 1, 2, \ldots\}$ denote the set of nonnegative integers, and for a positive integer L we define $[L] := \{0, 1, \ldots, L-1\}$. All logarithms have base 2 unless otherwise specified. Define $\chi(x) = \exp(2\pi i \cdot x)$ and observe that $\chi(x)\chi(y) = \chi(x+y)$.

2.1 CSIDH Group Action

Here we recall sufficient background on CSIDH for our purposes; for full details, see [CLM$^+$18]. At its heart is a free and transitive group action $\star \colon G \times Z \to Z$, where the group G is the ideal class group $\mathrm{Cl}(\mathcal{O})$ of the order $\mathcal{O} = \mathbb{Z}[\sqrt{-p}]$ of the imaginary quadratic number field $\mathbb{Q}(\sqrt{-p})$, for a given prime p of a certain form. (The acted-upon set Z is a certain collection of elliptic curves over $\mathbb{F}_p$, each of which can be uniquely represented by a single element of $\mathbb{F}_p$, but this will not be important for our purposes.) Because $\mathcal{O}$ is commutative, its class group $G = \mathrm{Cl}(\mathcal{O})$ is abelian. Heuristically, G is cyclic or "almost cyclic" (i.e., it has a cyclic component of order nearly as large as $|G|$), and its order $N = |G|$ is approximately $\sqrt{p}$.

CSIDH uses d special ideals $\mathfrak{l}_i$ of the order $\mathcal{O}$. Heuristically, these ideals generate the class group or a very large subgroup thereof; for simplicity, assume the former. The ideals $\mathfrak{l}_i$ define an integer lattice of relations

$$\Lambda = \{\mathbf{z} = (z_1, \ldots, z_d) \in \mathbb{Z}^d : \mathfrak{l}_1^{z_1} \cdots \mathfrak{l}_d^{z_d} \text{ is principal}\},$$

so G is isomorphic to $\mathbb{Z}^d/\Lambda$, via (the inverse of) the map $\mathbf{e} \in \mathbb{Z}^d \mapsto [\prod_i \mathfrak{l}_i^{e_i}]$, of which Λ is the kernel.

A CSIDH secret key is a vector $\mathbf{e} \in \mathbb{Z}^d$ of "small" integer exponents representing a group element; more specifically, the e_i are drawn uniformly from some small interval $[-B, B]$. One evaluates the group action for the associated ideal class $\left[\mathfrak{l}_1^{e_1} \cdots \mathfrak{l}_d^{e_d}\right]$ by successively applying the action of each $[\mathfrak{l}_i]$ or its inverse, $|e_i|$ times. Therefore, the ℓ_1 norm of $\mathbf{e}$ largely determines the evaluation time. Note that a group element is not uniquely specified by an exponent vector; any vector in the same coset of Λ defines the same group element, but very "long" vectors are not immediately useful for computing the group action. However, if we have a basis of Λ made up of very short vectors, then given any exponent representation of a group element, we can efficiently reduce it to a rather short representation of the same element using standard lattice algorithms like Babai's nearest-plane algorithm [Bab85].

In the CSIDH-512 parameterization, for which $p \approx 2^{512}$, the class group $G = \mathrm{Cl}(\mathcal{O})$ has recently been computed [BKV19]: it is isomorphic to the additive cyclic group $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$ of integers modulo

$$
\begin{aligned}
N = {} & 3 \cdot 37 \cdot 1407181 \cdot 51593604295295867744293584889 \\
& \cdot 31599414504681995853008278745587832204909 \approx 2^{257.1},
\end{aligned}
$$

and is in fact generated by the class of the ideal $\mathfrak{l}_1$. In addition, the lattice $\Lambda \subset \mathbb{Z}^{74}$ of relations among the ideals $\mathfrak{l}_i$ is known, along with a very high-quality (HKZ-reduced) basis. Indeed, the authors of [BKV19] showed that a uniformly random element of $\mathbb{Z}_N$ can be quickly reduced to a short exponent vector having a norm distribution very similar to the CSIDH-512 one. So, in summary, for CSIDH-512 we can efficiently represent the class group as $\mathbb{Z}_N$, and secret keys using the distinguished representatives $\{0, 1, \ldots, N-1\}$.

2.2 Abelian Hidden-Shift Problem

The hidden-shift problem on an additive abelian group G is as follows: given injective functions $f_0, f_1 \colon G \to X$ (for some arbitrary set X) such that $f_1(x) = f_0(x + s)$ for some secret "shift" $s \in G$ and all $x \in G$, the goal is to find s. For cyclic groups $G \cong \mathbb{Z}_N$, this hidden-shift problem is equivalent to the hidden-subgroup problem on the Nth dihedral group (which has order $2N$). Kuperberg [Kup03] gave the first nontrivial quantum algorithm for this problem, which uses subexponential $\exp(\sqrt{\log N})$ quantum time and space.

As observed by Childs, Jao, and Soukharev [CJS10], there is a simple connection between the abelian hidden-shift problem and the key-recovery problem for Couveignes–Rostovtsev–Stolbunov-type systems: given the "base value" $z_0 \in Z$ and a public key $z_1 = s \star z_0$ for some secret key $s \in G$, where $\star \colon G \times Z \to Z$ is a free and transitive group action, define $f_b \colon G \to Z$ as $f_b(g) = g \star z_b$ for $b = 0, 1$. These f_b are injective because $\star$ is free and transitive, and $f_1(x) = x \star z_1 = x \star (s \star z_0) = (x + s) \star z_0 = f_0(x + s)$, as required. So, solving the hidden-shift problem for these f_b immediately yields the secret key.

3 Collimation Sieve for Cyclic Groups

In this section we generalize Kuperberg's collimation sieve [Kup11] to arbitrary cyclic groups $\mathbb{Z}_N$ of known order N. (The algorithm can be made to work even if we just have a bound on the group order.) In essence, our algorithm works very much like Kuperberg's for power-of-two group orders $N = 2^n$, but with the following implementation differences and optimizations:

1. The sieve creates phase vectors having multipliers that lie in progressively smaller *intervals* of the integers. (Kuperberg instead makes the multipliers *divisible* by progressively larger powers of two.)
2. After sieving down to an interval of size S, where S can be roughly as large as the amount of quantumly accessible classical memory (QRACM), the algorithm applies a quantum Fourier transform

of dimension S and measures, to reveal about $\log S$ of the "most-significant bits" of the secret with good probability. (Kuperberg's algorithm instead applies a two-dimensional Fourier transform and measures to recover the *least*-significant bit of the secret with certainty.)

3. Alternatively, instead of recovering just $\log S$ bits of the secret, the algorithm can perform additional independent sieves down to various "scaled" intervals. By combining the resulting phase vectors, the algorithm can recover about $\log S$ different secret bits per sieve, and in particular, it can recover the entire secret using about $\log N / \log S = \log_S N$ sieves. (Kuperberg's algorithm, after recovering the least-significant bit of the secret, effectively halves the secret and repeats to recover the remaining bits, using $\log N$ total sieves.)

The technique from Item 2 is reminiscent of one used by Levieil and Fouque [LF06] to recover several secret bits at once in the Learning Parity with Noise problem. The technique in Item 3 is analogous to one attributed to Høyer in [Kup03] for recovering the entire secret from about $\log N$ qubits obtained via Kuperberg's original sieving technique.

3.1 Phase Vectors

We recall from [Kup11] the notion of a phase vector and some of its essential properties. Fix some positive integer N and $s \in \mathbb{Z}_N$. For a positive integer L, a *phase vector* of *length* L is a (pure) quantum state of the form

$$|\psi\rangle = L^{-1/2} \sum_{j \in [L]} \chi(b(j) \cdot s/N)|j\rangle$$

for some function $b \colon [L] \to \mathbb{N}$, where the $b(j)$ are called the *(phase) multipliers*. In all the algorithms considered in this work, the multiplier functions b will be written down explicitly in a table, in sorted order by $b(j)$ for efficiency of collimation (see Section 3.2). Note that while this requires classical memory proportional to L, only $\log L$ qubits of quantum memory are needed for $|\psi\rangle$. Also observe that the multipliers are implicitly modulo N, but we will mainly work with their distinguished integer representatives in $\{0, 1, \ldots, N-1\}$. We say that $|\psi\rangle$ is *ranged on* (or just *on*) a particular $S \subseteq \mathbb{Z}$ if every $b_j \in S$.

Looking ahead, a phase vector will be useful to us if its multiplier function b behaves roughly like a random function to some *small* desired set, e.g., a small interval $[S]$; see Section 3.4 for details. We will construct such phase vectors by recursively combining ones that have random-looking multiplier functions to somewhat bigger sets.

Creating and combining phase vectors. Prior (finite) hidden-subgroup and hidden-shift algorithms start from a simple quantum procedure (an "oracle") U_f that generates a special kind of one-qubit state, i.e., a length-2 phase vector. Given quantum procedures for computing injective functions $f_0, f_1 \colon \mathbb{Z}_N \to X$ (for an arbitrary set X) such that $f_1(x) = f_0(x+s)$ for some secret s and all x, the procedure U_f outputs a uniformly random $b \in \mathbb{Z}_N$ along with a qubit

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + \chi(b \cdot s/N)|1\rangle),$$

i.e., a length-2 phase vector with $b(0) = 0, b(1) = b$. The details of U_f are not material here; see [Kup03, Reg04] for accessible expositions.

Phase vectors can be naturally combined by tensoring: given phase r vectors $|\psi_i\rangle$ respectively having lengths L_i and multiplier functions b_i, we can form the following quantum state $|\psi'\rangle$ with index set $L =$

$[L_1] \times \cdots \times [L_r]$:

$$|\psi\rangle = |\psi_1, \ldots, \psi_r\rangle = |L|^{-1/2} \sum_{j_1 \in [L_1]} \cdots \sum_{j_r \in [L_r]} \chi(b_1(j_1) \cdot s/N) \cdots \chi(b_r(j_r) \cdot s/N) |j_1, \ldots, j_r\rangle \tag{3.1}$$
$$= |L|^{-1/2} \sum_{\vec{\jmath} \in L} \chi(b(\vec{\jmath}) \cdot s/N) |\vec{\jmath}\rangle,$$

where $b(\vec{\jmath}) = \sum_{i=1}^{r} b_i(j_i)$. Therefore, $|\psi\rangle$ can be thought of as a kind of phase vector of length $|L| = \prod_{i=1}^{r} L_i$, except that its index set is not exactly $[|L|]$ (although there is a natural bijection between L and $[|L|]$). More importantly, we will never explicitly write down the full multiplier function b, but will first "filter" $|\psi\rangle$ to have smaller length using a process called *collimation*, described next.

3.2 Collimating Phase Vectors

Algorithm 1 is our variant of Kuperberg's collimation procedure; the only significant difference is that it collimates according to "high bits" (or "middle bits"; see Section 3.2.3) rather than "low bits," which we use to deal with arbitrary group orders N. More precisely, it collimates phase vectors according to the quotients (ignoring remainder) of their multipliers with the desired interval size S, yielding a new phase vector on $[S]$.

In more detail, given phase vectors $|\psi_i\rangle$ having lengths L_i and multiplier functions $b_i \colon [L_i] \to \mathbb{Z}$, the algorithm constructs a combined phase vector $|\psi'\rangle$ having multiplier function $b'(\vec{\jmath}) = \sum_{i=1}^{r} b_i(j_i)$, as shown in Equation (3.1) above. It then measures the quotient $q = \lfloor b'(\vec{\jmath})/S \rfloor$, so that the "surviving" indices $\vec{\jmath}$ are exactly those for which $b'(\vec{\jmath}) \in qS + [S]$. The common additive qS term corresponds to a global phase that is easily removed (or ignored, since it has no measurable effect), so the surviving phase multipliers can be seen to lie in $[S]$. Let J be the set of surviving indices and suppose that $|J| = L$. Exactly as described in [Kup11], the algorithm constructs a bijection $\pi \colon J \to [L]$ (and its inverse) and applies a corresponding unitary permutation operator U_π to the (post-measurement) state, finally yielding a true length-L phase vector on $[S]$.

Algorithm 1 Collimate phase vectors.

Input: Phase vectors $|\psi_1\rangle, |\psi_2\rangle, \ldots, |\psi_r\rangle$ of respective lengths $L_1, \ldots, L_r$, and a desired interval size S.
Output: A phase vector $|\psi\rangle$ on $[S]$.

1. Form the phase vector $|\psi'\rangle = |\psi_1, \ldots, \psi_r\rangle$ having index set $[L_1] \times \cdots \times [L_r]$ and phase multiplier function $b'(\vec{\jmath}) = \sum_{i=1}^{r} b_i(j_i)$.
2. Measure $|\psi'\rangle$ according to the value of $q = \lfloor b'(\vec{\jmath})/S \rfloor$ to obtain $P_q|\psi'\rangle$ for a certain subunitary P_q.
3. Find the set J of tuples $\vec{\jmath}$ that satisfy the above. Let $L = |J|$ and choose a bijection $\pi \colon J \to [L]$.
4. Output phase vector $|\psi\rangle = U_\pi P_q |\psi'\rangle$ with index set $[L]$ and multiplier function $b(j) = b'(\pi^{-1}(j))$.

3.2.1 Length Analysis

Algorithm 1 is guaranteed to output a phase vector on $[S]$, but the length of the output is a random variable affected by the phase multipliers of the input vectors and the quantum measurement.

Let r be small, with $r = 2$ being the main case of interest. Suppose that the input vectors $|\psi_i\rangle$ have roughly uniformly distributed multipliers on $[S']$ for some $S' \gg S$, and let $L' = \prod_i L_i$ be the product of their lengths. Then the L' phase multipliers $b'(\vec{\jmath})$ are also very well distributed on $[rS']$ (though not quite uniformly, because extreme values are less likely). So, we expect $L \approx L' \cdot S/(rS')$ indices to "survive" collimation. (E.g., when $r = 2$, an easy calculation shows that $\mathbb{E}[L]$ is very close to $\frac{2}{3}L' \cdot S/S'$, due to the non-uniformity of pair sums.) Moreover, the surviving multipliers are close to uniformly distributed on $[S]$, because it is a very narrow subinterval of $[rS']$.

Because we will want all the input and output vectors to have roughly the same lengths L, we can therefore take $rS'L \approx SL'$ where $L' = L^r$, i.e.,

$$S' \approx S \cdot L^{r-1}/r. \tag{3.2}$$

In other words, with one level of collimation we can narrow the size of the interval in which the multipliers lie by roughly an L^{r-1}/r factor, while expecting to roughly preserve the vector lengths.

3.2.2 Complexity of Collimation

Kuperberg [Kup11, Proposition 4.2] rigorously bounds the complexity of his collimation procedure for the arity $r = 2$ of main interest. Letting $L_{\max}$ denote the maximum of the lengths of the input and output phase vectors and adopting our notation, Kuperberg proves that collimation can be done with:

- $\tilde{O}(L_{\max})$ classical time, where $\tilde{O}$ hides logarithmic factors in both $L_{\max}$ and N,
- $O(L_{\max} \log N)$ classical space,
- $O(L_{\max} \cdot \max(\log(S'/S), \log L_{\max}))$ bits of QRACM, and
- only $\mathrm{poly}(\log L_{\max})$ quantum time and $O(\log L_{\max})$ quantum space.

The same holds for our more general form of collimation, with one small subtlety concerning the QRACM usage. Naïvely, measuring $q = \lfloor b(\vec{\jmath})/S \rfloor$ requires storing the entire b_i vectors in QRACM, which requires up to $O(L_{\max} \log S') = O(L_{\max} \log N)$ bits. This is in contrast to Kuperberg's method, which requires only $O(L_{\max} \log(S'/S))$ bits, namely, the least-significant bits of the multipliers. We can obtain the latter bound by storing in QRACM only sufficiently many of the "most significant bits" of the $b_i(j_i)$, namely, $\hat{b}_i(j_i) = \lfloor b_i(j_i)/K \rfloor$ for some K moderately smaller than S. We then measure $q = \lfloor K \cdot \hat{b}(\vec{\jmath})/S \rfloor$, from which it follows that

$$\hat{b}(\vec{\jmath}) \in q(S/K) + [0, S/K] \Longrightarrow b(\vec{\jmath}) \in qS + [0, S + rK].$$

By taking, say, $K \geq (S/S')S$, we use at most $\log(S'/K) \leq 2\log(S'/S)$ bits per entry of $\hat{b_i}(j_i)$. And due to Equation (3.2), the range size for the collimated output vector is $S + rK \approx S(1 + r^2/L)$, which is insignificantly larger than S for the $L \geq 2^{16}$ of interest.

The $\tilde{O}(L_{\max})$ classical time bound is achieved by keeping the phase multipliers in a sorted table, which allows the *inverse* permutation $\pi^{-1} \colon [L] \to J$ to be classically computed and stored in an ordinary array. In addition, the forward permutation can be stored using an associative array in just $O(L_{\max} \log L_{\max})$ total bits. Quantum random access to these arrays allows U_π to be applied to the quantum state in quantum time polynomial in $\log L_{\max}$. See [Kup11, Section 4.3] for full details.

A close inspection shows that the constant factor in the QRACM bound is small: the entire algorithm can be run using four lookups (in superposition) into $L_{\max}$ indexable QRACM cells totaling $4L_{\max} \log L_{\max}$ bits or less. This is because:

- The two multiplier functions $\hat{b}_i$ described above can each be stored with $2\log(S'/S) \leq 2\log L \leq 2\log L_{\max}$ bits per entry, then this memory can be reused after the measurement. (Note that the factor of 2 here is not tight.)

- An associative array representing the permutation $\pi\colon J \to [L]$ can be stored with as little as $2\log L_{\max}$ bits per entry, by storing a map from each j_1 to the first value of j_2 such that $(j_1, j_2) \in J$ and the corresponding value of j; this memory also can be reused after applying the permutation operator U_1.

- An array storing the permutation $\pi^{-1}\colon [L] \to J$ can also be stored with $2\log L_{\max}$ bits per entry, and this memory also can be reused later, after applying the permutation operator U_2.

Finally, we remark that for the $L_{\max} \leq 2^{40}$ of interest in this work, the $\mathrm{poly}(\log L_{\max})$ quantum time and $O(\log L_{\max})$ quantum space needed for collimation (which involves just addition, division, and table lookups) are insignificant compared to the estimated complexity of implementing the quantum oracle U_f for CSIDH parameters of interest [BLMP19].

3.2.3 Scaled Intervals

Collimation naturally generalizes to produce phase vectors ranged on other sets, such as "scaled intervals" $A \cdot [S] = \{0, A, 2A, \ldots, (S-1)A\}$ for positive integers A. (We use such sets in Section 3.4.3 below.) Specifically, if we are given r phase vectors on $A \cdot [S']$, we can get a phase vector on certain scalings of $[S]$ as follows:

1. We can collimate according to $q = \lfloor b'(\vec{\jmath})/(AS) \rfloor$, thereby creating a phase vector on $A \cdot [S]$ (ignoring the global-phase term qAS), because all the $b(\vec{\jmath})$ are divisible by A.

2. Alternatively, we can collimate according to $c = b'(\vec{\jmath}) \bmod (AB)$ for $B = \lceil rS'/S \rceil$, thereby creating a phase vector on $AB \cdot [S]$ (ignoring the global-phase term c), because all the $b(\vec{\jmath})$ are in $A \cdot [rS']$.

3. Finally, we can interpolate between the above two techniques, collimating according to *both* $q = \lfloor b'(\vec{\jmath})/(ABS) \rfloor$ and $c = b'(\vec{\jmath}) \bmod (AB)$ for an arbitrary positive integer $B \leq \lceil rS'/S \rceil$, thereby creating a phase vector on $AB \cdot [S]$.

By appropriately composing these kinds of collimations, we can obtain any needed scaling factor. For all these options, adapting the above analyses yields the same ultimate conclusions, that collimation can decrease the range size by roughly an L^{r-1}/r factor while keeping the input and output vector lengths roughly equal.

3.3 Collimation Sieve

Algorithm 2 is the full collimation sieve, which constructs a phase vector on a desired interval $[S]$ by recursively constructing phase vectors on a suitably wider interval $[S']$, in a depth-first manner. The algorithm is essentially the same as Kuperberg's from [Kup11] (which incorporates a key insight of Regev's [Reg04]), except that it uses collimation on "high bits," along with a few tweaks to make it more effective in practice (see Section 3.3.1 below).

Typically, one would initially run Algorithm 2 with interval sizes S_i where:

- $S_0 \approx L$, the desired length of the ultimate phase vector (which can be almost as large as the available amount of QRACM), and
- $S_{i+1} = \min\{\approx S_i \cdot L^{r-1}/r, N\}$ (see Equation (3.2)), where the final $S_d = N$.

In the base case (when requesting a phase vector on $[N]$ of length $\approx L$), the algorithm simply invokes the oracle U_f some $\ell = \lfloor \log L \rceil$ times to get length-2 phase vectors $|\psi_i\rangle \propto |0\rangle + \chi(b_i \cdot s/N)$ for known uniformly random multipliers $b_i \in [N]$, then tensors them all together to get a length-2^ℓ phase vector whose multipliers are the mod-N subset-sums of the b_i. In the recursive case (when requesting a phase vector on $[S_i]$ for some $S_i < N$), the algorithm recursively obtains r phase vectors on $[S_{i+1}]$ of appropriate lengths, collimates them, and returns the result.

Naturally, we can sieve to other desired output ranges, like scaled intervals $S^i \cdot [S]$, simply by using the alternative kinds of collimations described in Section 3.2.3.

Algorithm 2 Collimation sieve for group $\mathbb{Z}_N$ and collimation arity r.

Input: Interval sizes $S_0 < S_1 < \cdots < S_d = N$, a desired phase-vector length L, and oracle access to U_f.
Output: A phase vector on $[S_0]$ of length $\approx L$.

Base case. If $S_0 = N$, generate $\ell = \lfloor \log L \rceil$ length-2 phase vectors $|\psi_1\rangle, |\psi_2\rangle, \ldots, |\psi_\ell\rangle$ using U_f (see Section 3.1). Output the length-2^ℓ phase vector $|\psi\rangle = |\psi_1, \ldots, \psi_\ell\rangle$.

Recursive case. Otherwise:

1. Using r recursive calls for sizes $S_1 < \cdots < S_d = N$ and appropriate desired lengths, obtain r phase vectors $|\psi_1\rangle, \ldots, |\psi_r\rangle$ on $[S_1]$, the product of whose lengths is $\approx L \cdot S_1/S_0$.
2. Collimate these phase vectors using Algorithm 1 to produce a phase vector $|\psi\rangle$ on $[S_0]$, and output it (unless its length is much less than L, in which case discard it and restart Step 2.)

3.3.1 Practical Improvements

Recall that the length of a phase vector output by collimation is unpredictable, and may be rather longer or shorter than expected. Because the lengths directly affect the required amount of QRACM and other resources required by the rest of the sieve, we would like to keep them under control as much as possible. We do so with two techniques:

1. *being adaptive* about the requested vector lengths in the recursive calls, and
2. *discarding* phase vectors that are unusually short, and recomputing from scratch.

Adaptivity means the following. Recall that to create a phase vector on $[S]$ of length $\approx L$, the algorithm recursively creates r phase vectors on $[S']$ for some given $S' \gg S$, the product of whose lengths we want to be $\approx L' = L \cdot (S'/S)$. So, on the first recursive call we request a vector of length $(L')^{1/r}$, and obtain a vector of some length $\tilde{L}$. Following that we want the product of the remaining $r-1$ vector lengths to be $\approx L'/\tilde{L}$, so we request a vector of length $(L'/\tilde{L})^{1/(r-1)}$, and so on. This immediately compensates for shorter-than-expected vectors, which helps to avoid cascading short vectors higher in the recursion tree and a useless final output. And in the fortunate event that we get a longer-than-expected vector, requesting correspondingly shorter vectors speeds up the remaining computation. It is also trivial to shorten a longer-than-expected vector via a partial measurement, in case we have a hard cap on the available amount of QRACM.

Vectors that are *much* shorter than expected present a more significant problem, however. Compensating for them requires corresponding longer vectors, which require correspondingly more QRACM and computation. And getting another short vector in that part of the computation subtree further increases the required resources. Therefore, whenever a call to Algorithm 2 produces a candidate output vector that is shorter than the requested length by some fixed threshold factor, it simply discards it and computes a fresh one from scratch.[5] Empirically, factors of 0.25 or 0.4 seem to work well, causing a discard in only about 2% or 4.5% of calls (respectively), and keeping the maximum vector length across the entire sieve to within a factor of about 2^4–2^5 or $2^{2.5}$ (respectively) of the desired length L; moreover, the factor tends to decrease somewhat as L grows. (See Figure 1 for details.) Without our discard rule, the maximum vector length tends to be several hundreds or even thousands of times larger than L, yet the final phase vector can still be much shorter than desired.

3.3.2 Complexity Model

For the main collimation arity $r = 2$ of interest, we give a model for the behavior of the collimation sieve. Based on the length analysis for $r = 2$ given in Section 3.2.1, we take $S_i = S_0 \cdot (2L/3)^i$ for all $i = 1, \dots, d-1$ such that $S_i < N$, and take $S_d = N$. So, the recursion depth of the sieve is given by

$$S_0 \cdot (2L/3)^d \geq N \implies d = \left\lceil \frac{\log(N/S_0)}{\log(2L/3)} \right\rceil.$$

At the leaf level of the recursion tree (i.e., in the base case), we typically need to make a phase vector of length about

$$L' = \sqrt{3LS_d/(2S_{d-1})} = \sqrt{3LN/(2S_{d-1})},$$

which we construct by making $\log L'$ (rounded up or down) queries to U_f and tensoring the results. Supposing that a random δ fraction of recursive calls to Algorithm 2 result in a discard (due to insufficient length), the arity of the recursion tree is effectively $2/(1-\delta)$. Therefore, the total number of queries to oracle U_f should be close to

$$Q = (2/(1-\delta))^d \cdot \log L'.$$

Our experiments turn out to conform very closely to this model, especially for moderate and larger values of L. See Section 4 for details.

3.4 Post-Processing

3.4.1 Regularization

An initial call to Algorithm 2 outputs a phase vector $|\psi\rangle$ on $[S] = [S_0]$ of length $\tilde{L} \approx L$, which we want to be somewhat larger than S. Heuristically, for each $t \in [S]$ we expect about $\tilde{L}/S$ phase multipliers $b(j)$ to equal t; however, there is some variation in the number of each multiplier. Ideally, we would like a *regular* state, i.e., one which has exactly the same number of multipliers for each $t \in [S]$.

We can obtain one by generalizing [Kup11]: select a maximal subset $X \subseteq [\tilde{L}]$ for which $b(X)$ has an equal number of every $t \in [S]$. Then measure whether $|\psi\rangle$ is in $\mathbb{C}[X]$, which holds with probability $|X|/\tilde{L}$. If not, discard it and run the sieve again; if so, the measured form of $|\psi\rangle$ is regular. It therefore has a factor of the form

$$S^{-1/2} \sum_{j \in [S]} \chi(j \cdot s/N) |j\rangle\,,$$

[5] This is roughly analogous to what is done in Kuperberg's original sieve [Kup03], where combining two qubits has a 50% chance of producing a "useless" output that is then discarded.

which we can extract by reindexing. (This requires almost no work, because the multipliers are sorted.) Observe that the above state essentially corresponds to the dimension-S inverse quantum Fourier transform of a point function at sS/N; see Section 3.4.4 for details.

The probability of obtaining a regular phase vector is $|X|/\tilde{L} = mS/\tilde{L}$, where m is the frequency of the least-frequent phase multiplier $t \in [S]$. In our experiments, a length $\tilde{L} \approx 64S$ typically led to success probabilities in the 40–80% range, and a length $\tilde{L} \approx 128S$ usually led to an 80% or larger success probability.

3.4.2 Punctured Regularization

The above procedure is somewhat wasteful, because it loses a factor of $\tilde{L}/S \approx 2^7$ in the number of basis states $|j\rangle$ in the fortunate case (and loses all of them in the unfortunate case). Alternatively, we can use the following method for generating a "punctured" (regular) phase vector, which works for S as large as $\tilde{L}$ (or even a bit more), and which produces a state that is almost as good as a regular one on $[S]$. Empirically, this lets us extract almost $\log S$ bits of the secret.

Again suppose that the sieve produces a phase vector $|\psi\rangle$ on $[S]$ of length $\tilde{L}$. We make a pass over $j \in [\tilde{L}]$, forming a set X of one index j for each distinct value of $b(j)$, and ignoring duplicates. (This is trivial to do, because the multipliers are sorted.) We then measure whether $|\psi\rangle$ is in $\mathbb{C}[X]$, which holds with probability $|X|/\tilde{L}$. If not, we try again with a new choice of X on the leftover phase vector, as long as it remains long enough. If so, the restriction $b\colon X \to [S]$ is injective, so by reindexing each $j \in X$ to $b(j)$, we now have a state of the form

$$|X|^{-1/2} \sum_{j \in X} \chi(b(j) \cdot s/N)|j\rangle = |X|^{-1/2} \sum_{j \in b(X)} \chi(j \cdot s/N)|j\rangle. \tag{3.3}$$

This state is a length-$|X|$ phase vector, except for the "punctured" index set $b(X) \subseteq [S]$. It is also almost as good as a regular phase vector on $[S]$, in the following sense. Heuristically, each of the multipliers $b(j)$ for $j \in [\tilde{L}]$ is uniformly random, so the multipliers $b(X) \subseteq [S]$ form a random subset of density

$$1 - (1 - 1/S)^{\tilde{L}} \approx 1 - (1 - 1/e)^{\tilde{L}/S}.$$

(For example, this density is approximately 0.368, 0.600, and 0.840 for $\tilde{L} = S$, $2S$, and $4S$, respectively.) Therefore, the state in Equation (3.3) corresponds to a kind of *densely subsampled* Fourier transform of a point function encoding the secret. Empirically, such states have enough information to let us extract about $\log S - 2$ bits of the secret in expectation; see Section 3.4.4 for details.

3.4.3 Combining (Punctured) Regular Phase Vectors

By combining k separately generated regular phase vectors for *scalings* of $[S]$, we can create a regular phase vector on $[T]$ for $T = S^k$, as shown below. In particular, for $k > \log_S N$ we can create a regular phase vector for $T > N$, which is large enough to recover s exactly (with good probability). Note that it might not be necessary to recover *all* of s in this manner; given partial information on s (say, half of its bits) it might be more efficient to use other methods to recover the rest.

We separately create k regular phase vectors

$$|\psi_i\rangle = S^{-1/2} \sum_{j \in [S]} \chi(S^i j \cdot s/N)|j\rangle$$

on the scaled intervals $S^i \cdot [S] = \{0, S^i, 2S^i, \ldots, (S-1)S^i\}$, for $i = 0, 1, \ldots, k-1$. Then their tensor product $|\psi\rangle = |\psi_0, \ldots, \psi_{k-1}\rangle$ is

$$|\psi\rangle = T^{-1/2} \sum_{j_0 \in [S]} \cdots \sum_{j_{k-1} \in [S]} \chi\Big(\sum_{i=0}^{k-1} j_i S^i \cdot s/N\Big) |j_0, \ldots, j_{k-1}\rangle = T^{-1/2} \sum_{j \in [T]} \chi(j \cdot s/N) |j\rangle,$$

where we have re-indexed using $j = \sum_{i=0}^{k-1} j_i S^i$. Therefore, $|\psi\rangle$ is a regular phase vector for $[T]$, as desired.

The same technique works for punctured regular states, where the tensored state's index set is the Cartesian product of the original states' index sets. To prevent the density from decreasing, we can use a scaling factor slightly smaller than S, e.g., δS where δ is the density of the input states. Then the density of the resulting state is about $(\delta S)^k/(\delta^{k-1} S^k) = \delta$.

3.4.4 Measurement

Now suppose we have a regular phase vector $|\psi\rangle = T^{-1/2} \sum_{j \in [T]} \chi(j \cdot s/N) |j\rangle$ on $[T]$. Then its T-dimensional quantum Fourier transform is

$$\mathrm{QFT}_T |\psi\rangle = T^{-1} \sum_{w \in [T]} \sum_{j \in [T]} \chi\left(\frac{js}{N} - \frac{jw}{T}\right) |w\rangle = T^{-1} \sum_{w} \left(\sum_{j} \chi\Big(j \Big(\frac{s}{N} - \frac{w}{T}\Big)\Big) \right) |w\rangle. \tag{3.4}$$

We compute this state and measure, obtaining some w that reveals information about s, as analyzed next.

If $N|(sT)$, then the amplitude associated with $w = sT/N \in [T]$ is nonzero and the amplitudes associated with all the other $w \in [T]$ are zero, so measuring the state yields w with certainty, from which we recover $s = wN/T$. Otherwise, fix some arbitrary $w \in [T]$ and let $\theta = s/N - w/T \notin \mathbb{Z}$. By summing the finite geometric series (over j), we see that the amplitude associated with $|w\rangle$ is

$$T^{-1} \left| \frac{1 - \chi(T\theta)}{1 - \chi(\theta)} \right| = T^{-1} \left| \frac{\chi(T\theta/2) \cdot (\chi(-T\theta/2) - \chi(T\theta/2))}{\chi(\theta/2) \cdot (\chi(-\theta/2) - \chi(\theta/2))} \right| = T^{-1} \left| \frac{\sin(\pi T \theta)}{\sin(\pi \theta)} \right|.$$

For $|\theta| \leq 1/(2T)$ this value is at least $(T \sin(\pi/(2T)))^{-1} \geq 2/\pi$. So when measuring the state, we obtain a w such that $|s/N - w/T| \leq 1/(2T)$ with probability at least $4/\pi^2 \geq 0.4$. In such a case, we have

$$s \in w \cdot \tfrac{N}{T} + [-\tfrac{N}{2T}, \tfrac{N}{2T}],$$

i.e., we know the $\log T$ "most-significant bits" of s. In particular, if $T > N$ then this defines s uniquely.

Now suppose instead that we have a *punctured* regular phase vector $|\psi\rangle = |Y|^{-1/2} \sum_{j \in Y} \chi(j \cdot s/N) |j\rangle$ on $[T]$, for a heuristically random index set $Y \subseteq [T]$ of significant density. Its QFT is exactly as in Equation (3.4), but with normalizing factor $(YT)^{-1/2}$ instead of T, and with the index j running over Y instead of $[T]$. As above, when w/T is very close to s/N, the amplitudes $\chi(j(s/N - w/T)) \in \mathbb{C}$ all point in roughly the same direction, and accumulate. Otherwise, the amplitudes heuristically point in random directions and mostly cancel out. Therefore, the final measurement is likely to output a w close to sT/N.

For the values of S we used in our experiments, it is efficient to compute the probability of obtaining any particular value of w when measuring (the QFT of) a particular punctured phase vector. Empirically, we usually observe a total probability (over the first several punctured vectors coming from the final sieve output) of about 40% or more in recovering the value of w closest to sT/N. This corresponds to extracting at least $\log T - 2$ bits of the secret in expectation. See Figure 2.

4 Experiments

At present, there are no (publicly available) quantum computers capable of running the full quantum algorithm for nontrivial parameters. But fortunately, as pointed out in [Kup11], the collimation sieve itself (apart from the quantum oracle U_f and the final QFT) is *pseudoclassical*: it consists entirely of permutations of the computational basis and measurements in that basis, which are trivial to simulate classically. In addition, the needed part of the quantum oracle U_f is easy to simulate, just by generating a uniformly random phase multiplier $b \leftarrow \mathbb{Z}_N$ (for the qubit $|\psi\rangle \propto |0\rangle + \chi(b \cdot s/N)|1\rangle$, which we do not need to generate).

4.1 Sieve Simulator

Using the above observations, we implemented a classical simulator for our generalized collimation sieve.[6] The simulator is currently hard-coded for collimation arity $r = 2$, but would be easy to generalize to larger arities. It allows the user to specify:

- a group order N (including an option for the exact CSIDH-512 group order, as computed in [BKV19]);
- a desired typical phase vector length L;
- an interval size S for the ultimate phase vector.

The simulator logs its progress in a human-readable form, and finally outputs various statistics for the full sieve, including:

- the total number $\tilde{Q}$ of queries to the quantum oracle U_f;
- the number Q of queries predicted by the model from Section 3.3.2;
- the length $\tilde{L}_{\max}$ of the longest created phase vector;
- the probability of obtaining a *regular* phase vector from the final one, and the expected number of bits of the secret that can be recovered from the final phase vector via regularity;
- the probabilities of obtaining *punctured* regular phase vectors of sufficient length from the final phase vector, and the total probability of measuring a value that yields $\log S$ secret bits.

4.2 Experimental Results

We ran our simulator for a wide range of group orders N (focusing mainly on the exact CSIDH-512 group order), desired phase-vector lengths L, and range sizes S. Our results for the CSIDH-512 group order are given in Figure 1 and Figure 2; the former concerns full regularization of the final phase vector (Section 3.4.1), while the latter concerns punctured regularization (Section 3.4.2). In summary, the experiments strongly support the following general conclusions:

- For all tested group orders and desired vector lengths $L \in [2^{16}, 2^{26}]$, the required *classical* resources are cryptanalytically insignificant: at most a few core-days on a commodity server with 128GB or 512GB of RAM, using only four CPU cores and less than 100GB RAM per experiment.

[6] The code for the simulator and instructions for running it are at `https://github.com/cpeikert/CollimationSieve`. The code is written in the author's favorite functional language Haskell, and has not been especially optimized for performance, but it suffices for the present purposes.

- The actual number $\tilde{Q}$ of oracle queries conforms very closely to the model from Section 3.3.2, especially for relatively larger $L \geq 2^{22}$, where $\tilde{Q}$ was almost always within a factor of $2^{0.4} \approx 1.32$ of the predicted Q, and was usually even closer.
- Taking $L = 64S$ suffices to obtain a *regular* phase vector on $[S]$ with good probability, usually in the 45–80% range. Halving S, and hence making $L \approx 128S$, typically results in a regularity probability of 70% or more, often yielding slightly more expected number of bits of the secret.
- Taking $L = S$ typically suffices to obtain at least $\log S - 2$ bits of the secret in expectation, via punctured regularization. More specifically, we can create one or more punctured regular phase vectors that collectively represent a roughly 40% probability of yielding $\log S$ bits of the secret.

5 Quantum (In)security of CSIDH

Here we come to some conclusions about the quantum security levels of various CSIDH parameterizations, based on our model from Section 3.3.2 and our experiments' close adherence to it. See Figure 3 for example estimates and Section 5.3 for details on how we arrived at them.

5.1 Complexity of Key Recovery

Our main conclusion is that CSIDH-512 key recovery can be accomplished using, for example, about 2^{16} oracle queries with about 2^{40} bits of QRACM; see Figure 3 for other workable options. This significantly improves the prior estimate [BS18] of about $2^{32.5}$ queries and 2^{31} *quantum* bits of memory, for a version of Kuperberg's original sieve algorithm [Kup03]. We stress that these query complexities are for recovering *almost all* of the bits of the secret. It seems likely, but remains to be seen, whether increasing the collimation arity and/or exploiting partial information about the secret can reduce the query complexity even further.

As shown below in Section 5.2, the QRACM can be implemented using purely classical memory and far fewer quantum resources than are needed to implement the oracle calls (under present estimates for the latter). Therefore, under the plausible assumption (see Section 1.4 for discussion) that implementing the quantum oracle is not much more costly than the "best conceivable" estimate of about 2^{40} nonlinear bit operations, which translates to between 2^{40} and 2^{44} T-gates [BLMP19], CSIDH-512 key recovery costs between roughly 2^{56} and 2^{60} T-gates (plus relatively small other resources); it would be prudent to expect that something toward the lower end of this range is attainable. In any case, this falsifies a standard interpretation of the claimed 64 bits of quantum security for CSIDH-512.[7]

NIST level 1 with MAXDEPTH. Under the same assumption about the quantum oracle, CSIDH-512 falls *well short* of the claimed NIST quantum security level 1—i.e., as hard as key search for AES-128—when accounting for the MAXDEPTH restriction. Specifically, NIST's estimate for breaking AES-128 is $2^{170}/\text{MAXDEPTH}$ quantum gates, where plausible values of MAXDEPTH range between 2^{40} and 2^{96}. As shown below in Section 5.2, for parameters of interest the quantum complexity of the collimation sieve itself (separate from the oracle implementation) is near the low end of the MAXDEPTH range. Moreover, the

[7]Note that [CLM+18] does not explicitly state the metric by which its quantum security claim is to be judged, but T-gate complexity is commonly used, including in [BLMP19], because T-gates appear much more expensive to implement than Clifford gates. Other metrics like "depth times width" can be considered, but in any reasonable metric the oracle calls are presently the bottleneck. We leave a detailed analysis under other metrics to future work.

$\log \tilde{Q}$	$\log Q$	$\log \tilde{L}_{\max}$	$\log L$	$\log S$	$\Pr$[regular] (%)	bits	threshold	discard (%)	depth
19.4	19.1	23.9	18	10	78	7.8	0.25	2.8	15
19.4	19.2	23.8		11	95	10.5		3.6	
19.2	19.3	23.3		12	72	8.6		4.2	
18.3	18.2	24.3	19	11	95	10.5		2.3	14
18.4	18.1	23.5		12	82	9.8		2.3	
18.6	18.1	24.5		13	61	7.9		2.4	
17.6	17.4	24.3	20	12	84	10.1		2.0	13
17.7	17.4	25.2		13	56	7.3		2.0	
17.6	17.4	24.2		14	66	9.2		2.2	
17.2	16.7	25.2	21	13	64	8.3		2.1	12
17.2	16.7	25.7		14	71	10.0		2.0	
16.8	16.6	25.4		15	73	10.9		1.9	
16.6	16.3	26.8	22	14	72	10.0		2.0	12
16.3	16.2	26.6		15	55	8.2		1.9	
16.6	16.2	26.6		16	60	9.6		2.3	
16.3	15.7	26.4	23	15	79	11.9		2.0	11
15.6	15.6	26.9		16	66	10.5		1.8	
15.6	15.6	26.7		17	62	10.6		2.0	
15.4	15.4	28.0	24	16	71	11.3		2.4	11
15.5	15.3	28.6		17	85	14.4		2.1	
15.3	15.2	29.1		18	64	11.5		2.1	
14.9	14.8	28.7	25	17	62	10.5		1.8	10
14.8	14.8	29.6		17	93	15.7		1.9	
15.4	14.8	28.9		18	85	15.3		1.9	
14.9	14.8	29.2		19	60	11.4		2.1	
15.1	14.8	29.1		19	81	15.4		2.0	
15.0	14.7	29.6	26	18	92	16.5	0.40	3.5	10
15.3	14.8	29.3		18	88	15.8		4.1	
14.9	14.8	29.4		19	77	14.7		4.6	

Figure 1: Statistics from representative runs of our collimation sieve simulator on the actual CSIDH-512 group, as computed by [BKV19]. Here $\tilde{Q}$ and Q are respectively the actual and predicted (by the model of Section 3.3.2) number of queries to the quantum oracle; $\tilde{L}_{\max}$ is the maximum length of all created phase vectors, and L is the requested (and typical) vector length; S is the range size for the final phase vector; "$\Pr$[regular]" is the probability of obtaining a regular vector from the final phase vector (see Section 3.4.1); "bits" is the expected number of bits of the secret that can be recovered from the final phase vector; "threshold" is the threshold factor used for determining whether a phase vector is too short (see Section 3.3.1); "discard" is the fraction of recursive calls that were discarded for being below the threshold; "depth" is the recursion depth of the sieve. Each missing entry is equal to the one above it. Every experiment ran on at most four CPU cores on a commodity server, and completed in no more than a few core-days.

$\log \tilde{Q}$	$\log Q$	$\log \tilde{L}_{\max}$	$\log L$	$\log S$	bits	threshold	discard (%)	depth
17.2	17.0	25.3	20	20	19.1	0.25	2.2	13
18.1	17.1	24.3			18.1		2.2	
16.7	16.4	25.1	21	21	20.1		2.0	12
16.8	16.4	24.9			19.4		1.9	
16.6	16.4	24.8			17.7		2.0	
15.9	15.7	26.6	22	22	21.2		1.9	11
16.2	15.8	25.7			20.3		2.0	
16.4	15.8	25.8			20.4		2.0	
15.6	15.3	26.6	23	23	21.2		1.7	11
16.0	15.4	26.3			21.9		2.0	
15.9	15.4	26.7			21.3		1.9	
16.1	15.4	26.1			21.4		1.9	
14.9	14.8	26.8	24	24	22.5		1.8	10
15.6	14.8	27.3			23.2		1.9	
15.0	14.8	27.3			23.1		1.9	
15.0	14.6	28.3	25	25	22.4		2.3	10
14.5	14.5	27.9			23.5		1.6	
15.0	14.6	28.2			23.7		2.4	
14.5	14.3	29.1	26	26	25.2		3.0	10
14.7	14.2	29.0			24.1		2.4	
14.7	14.6	28.4			25.0	0.40	4.9	
14.2	14.1	29.6	27	27	25.7		4.1	9
14.5	14.1	29.6			25.3		4.6	
14.4	14.1	30.0			24.6		4.2	

Figure 2: Statistics from representative runs of our collimation sieve simulator on the actual CSIDH-512 group, as computed by [BKV19]. The column headers are the same as in Figure 1, except that "bits" b is the expected number of secret bits obtainable by using punctured phase vectors obtained from the vector output by the sieve; see Section 3.4.2 and Section 3.4.4. Each missing entry is equal to the one above it. Every experiment ran on at most four CPU cores on a commodity server, and completed within several core-days.

oracle implementation's depth is less than its gate count, so the total depth of the CSIDH-512 attack is either near the low end of the MAXDEPTH range or only somewhat larger, depending on whether we make the oracle calls in parallel or sequentially (which would respectively require more or fewer qubits). In either case, the quantum gate complexity of the attack is far below 2^{130}, and even well below 2^{74} for the high end of MAXDEPTH.

Other CSIDH parameters. Similarly, key recovery for CSIDH-1024 and -1792 using the same or somewhat more QRACM requires only 2^b oracle queries, for b in the mid-20s and high-30s, respectively. (See Figure 3.) Although precise estimates for implementing the corresponding oracles are not yet available, these query complexities imply that these CSIDH parameterizations have at best moderate quantum security beyond what is given by the cost of implementing their oracles. Moreover, a similar analysis to the one above applies regarding CSIDH-1792's claimed NIST quantum security level 3—i.e., as hard as key search for AES-192—for which attacks are supposed to require $2^{233}/\text{MAXDEPTH}$ quantum gates.

5.2 Quantum Complexity of the Sieve

Here we estimate the T-gate complexity of the quantum work of the collimation sieve using the QRACM implementation of [BGB+18], as suggested by Schanck [Sch19]. The bottom line is that for parameters of interest, the quantum complexity of the sieve and the QRACM is dwarfed by that of the oracle calls (under current estimates for the latter).

Fix the collimation arity $r = 2$. The analysis below shows that the total T-gate complexity of the collimation sieve (apart from the oracle calls) is essentially

$$16\tilde{L} \cdot (2/(1-\delta))^d,$$

where $\tilde{L}$ is (an upper bound on) the typical phase-vector length, δ is the discard probability, and d is the recursion depth of the sieve. For $\delta \approx 0.02$ and sieve parameters $(\tilde{L}, d) \in \{(23, 11), (27, 10), (31, 8), (35, 7)\}$ (see Figure 3), this T-gate complexity estimate is between 2^{38} and 2^{47}, which is well below the existing lower bound of about 2^{53} for making at least 2^{14} calls to an oracle implementation with T-gate complexity at least 2^{39} [BLMP19].

The full sieve is a depth-first search of a binary tree (modulo discards), with collimation at each internal (i.e., non-leaf) node, and one or more oracle calls at each leaf node. Therefore, the T-gate complexity of the sieve itself (apart from the oracle calls) is essentially the number of internal nodes times the T-gate complexity of collimation. For sieve recursion depth d, the number of internal nodes is about 2^d, or more precisely $(2/(1-\delta))^d$ when accounting for discards.

The T-gate complexity of a single collimation step can be bounded as follows. For input and output phase vectors having lengths bounded by D, the quantum work is just four lookups (in superposition) into a QRACM of D indexable cells: one for each of the two input multiplier functions b_i, one for looking up a value of $\pi^{-1}(j)$, and one for looking up a value of $\pi(j_1, j_2)$. (The latter can be done using a table that maps j_1 to the first corresponding value of j_2 and the matching j, and adding the difference with the desired value of j_2.) Because [BGB+18] implements a QRACM of D cells (storing any number of bits) using classical memory plus just $4D$ T-gates (and only $\lceil \log D \rceil$ ancillary qubits), the claim follows.

5.3 Details of the Estimates

The estimates in Figure 3 are based on the following:

$\log p$	$\log N$	$\log L$	$\log$ QRACM	depth	$\log \tilde{T}$
512	257.1	22.3	32	11	19.3
		26.1	36	10	17.5
		29.9	40	8	16.1
		33.8	44	7	15.1
		37.6	48	6	14.2
1024	512	26.1	36	20	29.0
		29.9	40	17	26.3
		33.8	44	15	24.2
		37.6	48	13	22.5
		41.5	52	12	21.2
1792	896	29.9	40	30	40.7
		33.8	44	26	36.9
		37.6	48	24	34.1
		41.5	52	21	31.7
		45.4	56	19	29.7

Figure 3: Example complexity estimates for secret-key recovery against CSIDH-$\log p$ using the collimation sieve with arity $r = 2$, for various bit lengths (rounded to the nearest integer) of the CSIDH parameter p. Each missing entry is equal to the one above it. Here N is the estimated (or exact, in the case of CSIDH-512) group order; $L = S$ are respectively the desired length and range size of the sieve's final phase vector; "QRACM" is the number of bits of quantumly accessible classical memory, which is bounded by $4\tilde{L}_{\max} \log \tilde{L}_{\max}$ for $\tilde{L}_{\max} = 8L$ indexable cells (see Section 3.2.2); "depth" is the depth of the sieve's recursion tree, which is a proxy for (the logarithm of) one main factor in the classical running time; $\tilde{T}$ is the total number of queries to the quantum oracle to recover all but 56 bits of the secret.

- We take $S = L$ and use punctured regularity to obtain several bits of the secret (see Section 3.4.2). We assume that each run of the sieve reveals an expected $\log S - 2$ bits of the secret, which is consistent with our experiments.
- We quantify the total number $\tilde{T}$ of quantum-oracle queries needed to recover all but 56 bits of the secret; the remainder can be obtained by classical brute force.[8] We assume that the actual number of queries $\tilde{Q}$ made by a run of the sieve is within a factor of $2^{0.3}$ of the estimated number Q in expectation, which is consistent with our experiments.
- We impose a maximum phase-vector length of $\tilde{L}_{\max} = 8L$. This reflects the fact that the generated phase vectors are sometimes longer than the desired length L, but are almost always within a factor of 8, and we can enforce this as a hard bound by doing a partial measurement whenever a phase vector happens to be longer. We use a bound of $4\tilde{L}_{\max} \log \tilde{L}_{\max}$ on the number of bits of QRACM (see Section 3.2.2), though the number $\tilde{L}_{\max}$ of indexable cells is more relevant for costing.

References

[Bab85] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986. Preliminary version in STACS 1985.

[BGB+18] R. Babbush, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, A. Paler, A. Fowler, and H. Neven. Encoding electronic spectra in quantum circuits with linear T complexity. *Phys. Rev. X*, 8:041015, Oct 2018. `https://arxiv.org/pdf/1805.03662.pdf`.

[BHT98] G. Brassard, P. Høyer, and A. Tapp. Quantum cryptanalysis of hash and claw-free functions. In *LATIN*, pages 163–169. 1998.

[BIJJ18] J. Biasse, A. Iezzi, and M. J. Jacobson Jr. A note on the security of CSIDH. In *INDOCRYPT*, pages 153–168. 2018.

[BKV19] W. Beullens, T. Kleinjung, and F. Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. Cryptology ePrint Archive, Report 2019/498, 2019. `https://eprint.iacr.org/2019/498`.

[BLMP19] D. J. Bernstein, T. Lange, C. Martindale, and L. Panny. Quantum circuits for the CSIDH: Optimizing quantum evaluation of isogenies. In *EUROCRYPT*, pages 409–441. 2019.

[BS18] X. Bonnetain and A. Schrottenloher. Quantum security analysis of CSIDH and ordinary isogeny-based schemes. Cryptology ePrint Archive, Report 2018/537, 2018. `https://eprint.iacr.org/2018/537`.

[CJS10] A. M. Childs, D. Jao, and V. Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Mathematical Cryptology*, 8(1):1–29, 2014. `https://arxiv.org/abs/1012.4019`.

[CLM+18] W. Castryck, T. Lange, C. Martindale, L. Panny, and J. Renes. CSIDH: An efficient post-quantum commutative group action. In *ASIACRYPT*, pages 395–427. 2018.

[8]This choice seems conservative; it is plausible that classical attacks could relatively efficiently recover the full secret from, say, half of its bits or even fewer, which would cut the estimated number of quantum queries almost in half.

[Cou06] J.-M. Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. https://eprint.iacr.org/2006/291.

[DG19] L. De Feo and S. D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In *EUROCRYPT*, pages 759–789. 2019.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[JD11] D. Jao and L. De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography*, pages 19–34. 2011.

[JS19] S. Jaques and J. M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. Cryptology ePrint Archive, Report 2019/103, 2019. https://eprint.iacr.org/2019/103.

[Kup03] G. Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM J. Comput.*, 35(1):170–188, 2005. Preliminary version in https://arxiv.org/abs/quant-ph/0302112.

[Kup11] G. 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*, pages 20–34. 2013. Preliminary version in https://arxiv.org/abs/1112.3333.

[LF06] É. Levieil and P. Fouque. An improved LPN algorithm. In *SCN*, pages 348–359. 2006.

[NIS] NIST post-quantum cryptography project. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography.

[Reg04] O. Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. *CoRR*, quant-ph/0406151, 2004.

[RS06] A. Rostovtsev and A. Stolbunov. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. https://eprint.iacr.org/2006/145.

[Sch19] J. Schanck. Personal communication, June 2019.

[Sho94] P. 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 2004.

[SS79] R. Schroeppel and A. Shamir. A $t=o(2^{n/2})$, $s=o(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981. Preliminary version in FOCS 1979.

[Sto04] A. Stolbunov. *Public-key encryption based on cycles of isogenous elliptic curves*. Master's thesis, Saint-Petersburg State Polytechnical University, 2004. In Russian.

[Sto11] A. Stolbunov. *Cryptographic Schemes Based on Isogenies*. Ph.D. thesis, Norwegian University of Science and Technology, 2011.