Improved Low-qubit Hidden Shift Algorithms

Xavier Bonnetain

Improved Low-qubit Hidden Shift Algorithms

Xavier Bonnetain

[1] Sorbonne Université, Collège Doctoral, F-75005 Paris, France
[2] Inria, France

Abstract. Hidden shift problems are relevant to assess the quantum security of various cryptographic constructs. Multiple quantum subexponential time algorithms have been proposed. In this paper, we propose some improvements on a polynomial quantum memory algorithm proposed by Childs, Jao and Soukharev in 2010. We use subset-sum algorithms to significantly reduce its complexity. We also propose new tradeoffs between quantum queries, classical time and classical memory to solve this problem.

1 Introduction

The hidden shift problem can be stated as follows:

Let f, g be two injective functions, $\mathbb{G}$ a group. Given the promise that there exists $s \in \mathbb{G}$ such that, for all x, $f(x) = g(xs)$, retrieve s.

This is a generalization of the the hidden subgroup problem, which corresponds to the case $f = g$, and is efficiently solved by Shor's algorithm [16] in the abelian case.

As the hidden subgroup problem, the hidden shift problem is of interest for cryptography. Notably, the security of multiple symmetric primitives [1,4] and the security of some post-quantum isogeny-based asymmetric schemes [6,7,9,10] depends on its hardness.

It is also interesting for quantum computing, as solving this problem requires exponential classical time, but depending on the group structure, can be solved quantumly in either polynomial, sub-exponential or exponential time.

The first subexponential quantum algorithm for the hidden shift problem has been proposed by Kuperberg in [12], where he proposed multiple algorithms in $2^{O(\sqrt{n})}$ quantum time, memory and query. A polynomial quantum memory variant has been proposed by Regev in [14], with a time cost in $2^{O(\sqrt{n\log_2(n)})}$. This latter algorithm has been generalized and more precisely studied by Childs, Jao and Soukharev in [7], where they prove a time cost in $2^{(\sqrt{2}+o(1))\sqrt{n\log_2(n)}}$ for all abelian groups. In 2013, Kuperberg proposed a generalisation of his first algorithm and Regev's variant [13] with a heuristic cost estimate of $\widetilde{O}\left(2^{\sqrt{2n}}\right)$.

Asymptotic cost estimates provide little information on the concrete cost of the algorithm. Concrete cost estimates for groups of order a power of two for Kuperberg's original algorithm have been done in [4], obtaining a cost of around $2^{1.8\sqrt{n}}$ for the cyclic case. The used algorithm has later been generalised to arbitrary cyclic groups in [5], for a cost of around $12 \times 2^{1.8\sqrt{n}}$.

In this paper, we will focus on the quantum algorithm of Childs, Jao and Soukharev [7] and show how it can be improved.

1.1 Notations

We note

$$L_n(\alpha, c) = 2^{\left((c+o(1))n^{\alpha}\log_2(n)^{1-\alpha}\right)}.$$

We will focus here on the cyclic group $\mathbb{Z}/(N\mathbb{Z})$, and denote $n = \log_2(N)$. However, the algorithms can also be applied to arbitrary abelian groups, using the same approach as in [7]. We denote $L_n(1/2, c)$ as $L(c)$. We note $\cdot$ the inner product of vectors.

We denote $\chi(x) = \exp(2i\pi x)$ in the qubit phases.

1.2 Results

In this paper, we improve the quantum hidden shift algorithm of [7]. We use subset-sum algorithms to lower the exponent of the complexity, and propose different tradeoffs between quantum and classical cost. In particular, with c the exponent to solve a subset-sum instance, we solve the hidden shift problem in $L(\sqrt{c})$ classical time and quantum queries, and if we want a quadratic gap between the classical time cost and the quantum query cost, we can solve it in $L(\sqrt{c/3})$ quantum queries and $L(2\sqrt{c/3})$ classical time.

The results are summarized in Table 1. The quantum time is roughly the number of quantum queries, as the only non-trivial quantum operation is the oracle evaluation.

Quantum query	Classical time	Classical memory	Subset-sum	Source
$L\left(1/\sqrt{2}\right)$	$L\left(\sqrt{2}\right)$	$\widetilde{O}(1)$	exhaustive search	[7]
$L(1)$	$L(1)$	$\widetilde{O}(1)$	exhaustive search	Section 4.2
$L\left(1/\sqrt{3}\right)$	$L\left(2/\sqrt{3}\right)$	$\widetilde{O}(1)$	exhaustive search	Section 4.3
$L(0.539)$	$L(0.539)$	$L(0.539)$	[2]	Section 4.2
$L(0.312)$	$L(0.623)$	$L(0.312)$	[2]	Section 4.3
$L(0.849)$	$L(0.849)$	$\widetilde{O}(1)$	[2], poly.memory	Section 4.2
$L(0.490)$	$L(0.980)$	$\widetilde{O}(1)$	[2], poly.memory	Section 4.3
$O(n^2)$	$\widetilde{O}\left(2^{0.291n}\right)$	$\widetilde{O}\left(2^{0.291n}\right)$	[2]	Section 4.4

Table 1: Hidden shift algorithm cost

2 Hidden Shift Algorithms

Hidden shift algorithms are in two steps: the first one uses the oracles to produce some random qubits with a specific structure (the elements), which are then refined by some combination routines until we manage to extract the value of the hidden shift.

2.1 Element Generation

Given a quantum oracle access to f and g, one can compute

$$\sum_x |0\rangle |x\rangle |f(x)\rangle + |1\rangle |x\rangle |g(x)\rangle .$$

After a measurement of the last register, one obtains

$$|0\rangle |x_0\rangle + |1\rangle |x_0 + s\rangle$$

for a given unknown x_0. Now, one can apply a Quantum Fourier Transform on the second register, to obtain

$$\sum_\ell |0\rangle \chi\left(\frac{x_0\ell}{N}\right) |\ell\rangle + |1\rangle \chi\left(\frac{(x_0+s)\ell}{N}\right) |\ell\rangle .$$

Finally, one can perform a measurement on the second register, and obtain ℓ and

$$|\psi_\ell\rangle = |0\rangle + \chi\left(\frac{s\ell}{N}\right) |1\rangle .$$

2.2 Interesting Elements

If one manages to obtain $|\psi_1\rangle, \ldots, |\psi_{2^j}\rangle, |\psi_{2^n}\rangle$, then s can be retrieved with a Quantum Fourier Transform, as

$$\bigotimes_{j=0}^{n} |\psi_{2^j}\rangle = \sum_{\ell=0}^{2^n} \chi\left(\frac{s\ell}{N}\right) |\ell\rangle .$$

Hence, applying an inverse Quantum Fourier Transform allows to retrieve s.

It is to be noted that, for groups of odd order, we only need to be able to construct $|\psi_1\rangle$: the value 2^j can be obtained by multiplying all the labels by 2^{-j} and constructing 1 from the new labels.

The situation is slightly easier if $N = 2^n$. In that case, $|\psi_{2^{n-1}}\rangle = |0\rangle + (-1)^s |1\rangle$ directly gives $s \mod 2$. Likewise, as noted in [4], both $|\psi_{2^{n-2}}\rangle$ and $|\psi_{3\times 2^{n-2}}\rangle$ allows to obtain the second bit of s if $s \mod 2$ is known, and so on, knowing the lower bits of s allows to extract the next bit from $\left|\psi_{(2\alpha+1)2^j}\right\rangle$.

2.3 Combination routines

We will use the combination routines of [7] to obtain the labels we are looking for, that is, either $|\psi_1\rangle$ or $|\psi_{2^{n-1}}\rangle$.

The idea is to take a certain amount of elements (k), and use them to produce one better element. Recall that the elements are of the form $|0\rangle + \exp\left(2i\pi\frac{s\ell_i}{N}\right)|1\rangle$. If we tensor them, we obtain

$$\bigotimes_i |\psi_{\ell_i}\rangle = \sum_{j\in\{0,1\}^k} \chi\left(\frac{j\cdot(\ell_1,\dots,\ell_k)}{N}\right)|j\rangle .$$

Now, the objective of the combination routine is to perform a partial measurement on $|j\rangle$, in order to ensure that the remaining j have an interesting phase difference. After that, we only need to know the corresponding j, project the state on a pair, and relabel the two values of the pair to 0 and 1 in order to obtain a better element.

Algorithm 1 computes the function $|j\rangle|0\rangle \mapsto |j\rangle|j\cdot(\ell_1,\dots,\ell_k) \mod 2^r\rangle$ and measures the second register. By definition, the remaining j have a phase identical modulo 2^r. Hence, the output label will be a multiple of 2^r. This approach can be iterated, in order to obtain multiples of increasingly big powers of 2, and allows to reach $|\psi_{2^{n-1}}\rangle$.

Another approach to reach a specific element is to compute $|\lfloor j\cdot(\ell_1,\dots,\ell_k)/M\rfloor\rangle$. This will produce elements with a close phase (their difference will be smaller than M). As before, this approach can be iterated to obtain smaller and smaller labels, until we reach $|\psi_1\rangle$. This is done in Algorithm 2.

Algorithm 1 Combination routine, for powers of 2

```
Input: $(|\psi_{\ell_1}\rangle,\dots,|\psi_{\ell_k}\rangle) : \forall i, 2^a|\ell_i$, $r$
Output: $|\psi_{\ell'}\rangle$, $2^{r+a}|\ell'$
Tensor $\bigotimes_i |\psi_{\ell_i}\rangle = \sum_{j\in\{0,1\}^k}\chi\left(\frac{j\cdot(\ell_1,\dots,\ell_k)}{N}\right)|j\rangle$
Add an ancilla register, apply $|x\rangle|0\rangle \mapsto |x\rangle|x\cdot(\ell_1,\dots,\ell_k) \mod 2^r\rangle$
Measure the ancilla register, leaving with
    $V$ and $\sum_{j\cdot(\ell_1,\dots,\ell_k) \mod 2^r=V} \chi\left(\frac{j\cdot(\ell_1,\dots,\ell_k)}{N}\right)|j\rangle$
Compute the corresponding $j$
Pair them, project to a pair $(j_1,j_2)$.
    The register is now $\chi\left(\frac{j_1\cdot(\ell_1,\dots,\ell_k)}{N}\right)|j_1\rangle + \chi\left(\frac{j_2\cdot(\ell_1,\dots,\ell_k)}{N}\right)|j_2\rangle$
Map $|j_1\rangle$ to $|0\rangle$, $|j_2\rangle$ to $|1\rangle$
Return $|0\rangle + \chi\left(\frac{(j_2-j_1)\cdot(\ell_1,\dots,\ell_k)}{N}\right)|1\rangle$
```

Both algorithms are used in [7], the former to tackle cyclic groups of order a power of 2, the latter for cyclic groups of odd order.

Algorithm 2 Combination routine, for smaller labels

```
    **Input:** $(|\psi_{\ell_1}\rangle, \dots, |\psi_{\ell_k}\rangle), (\ell_1, \dots, \ell_k) \in [0; B)^k$, $r$
    **Output:** $|\psi_{\ell'}\rangle$, $\ell' < \sum_j \ell_j / 2^r$
1: Tensor $\bigotimes_i |\psi_{\ell_i}\rangle = \sum_{j \in \{0,1\}^k} \chi\left(\frac{j \cdot (\ell_1, \dots, \ell_k)}{N}\right) |j\rangle$
2: Add an ancilla register, apply $|x\rangle |0\rangle \mapsto |x\rangle \left|\lfloor x \cdot (\ell_1, \dots, \ell_k) 2^{r-1}/B \rfloor\right\rangle$
3: Measure the ancilla register, leaving with
       $$V \text{ and } \sum_{\lfloor j \cdot (\ell_1, \dots, \ell_k) 2^{r-1}/B \rfloor = V} \chi\left(\frac{j \cdot (\ell_1, \dots, \ell_k)}{N}\right) |j\rangle$$
4: Compute the corresponding $j$
5: Pair them, project to a pair $(j_1, j_2)$.
   The register is now $\chi\left(\frac{j_1 \cdot (\ell_1, \dots, \ell_k)}{N}\right) |j_1\rangle + \chi\left(\frac{j_2 \cdot (\ell_1, \dots, \ell_k)}{N}\right) |j_2\rangle$
6: Map $|j_1\rangle$ to $|0\rangle$, $|j_2\rangle$ to $|1\rangle$
7: Return $|0\rangle + \chi\left(\frac{(j_2 - j_1) \cdot (\ell_1, \dots, \ell_k)}{N}\right) |1\rangle$
```

Algorithm 3 Projection routine

```
Input: $\sum_{x \in J} \phi(x) |x\rangle$, $(j_1, j_2) \subset J$.
Output: $\phi(j_1) |j_1\rangle + \phi(j_2) |j_2\rangle$ or $\sum_{x \in J \setminus \{j_1, j_2\}} \phi(x) |x\rangle$
Add an ancilla qubit: $\sum_{x \in J} \phi(x) |x\rangle |0\rangle$
Apply the operator $|x\rangle |0\rangle \mapsto |x\rangle |x = j_1 \vee x = j_2\rangle$
Measure the ancilla qubit
```

Algorithm 3 projects on $\langle |j_1\rangle, |j_2\rangle \rangle$ with probability $2/|J|$, and otherwise projects to the supplementary vector space.

Finding pairs Finding the pairs for Algorithm 1 (Step 4) consists in finding the solutions of the equation $x \cdot (\ell_1, \dots, \ell_k) \mod 2^r = V$. This is addressed by brute-force in [7], for a cost of 2^k. For the complexity analysis, we consider that this step costs $\widetilde{O}(2^{ck})$, the brute-force case being the case $c = 1$. This brute-force approach can also be applied to Step 4 of Algorithm 2.

Complexity Algorithm 1 succeeds if, at step 5, we manage to project on a pair (j_1, j_2). Indeed, as they are both preimages of V, we have

$$\ell' = (j_2 - j_1) \cdot (\ell_1, \dots, \ell_k) = 0 \mod 2^m.$$

In order to achieve this, we must:

1. Have at least two distinct solutions of the equation,
2. Manage to find the solutions,
3. Successfully project onto a pair.

The first point requires us to have $r < k$. In [14], $k = r + 4$ is used, while [7] uses $k = r + 1$. In both cases, we have a fixed probability to have at least 2 solutions: there are 2^r images, hence at most 2^r subsets have a sum for which there is a unique preimage, hence we have at least 2 solutions at least half the time. The third point is a problem when the number of solutions is odd: in that case, we may fail to project on a pair. As we have balanced superpositions, the probability of obtaining a singleton is half the probability of obtaining a pair. Hence, we fail to project with probability at most 1/3.

The case of Algorithm 2 is very similar. In [7], the authors chose $r = k - \log(k)$. As the output labels depend on the sum of the inputs, they can be larger, and the output space is of size $k2^r$. Hence, in [7], a value of $r = k - \log(k)$ is chosen, to guarantee a constant success probability. Moreover, a step of rejection sampling (still with constant probability) is added, to guarantee a uniform sampling in the smaller interval.

To summarize, the two routines take k elements, and produce with constant probability p one refined element which is $r \simeq k$ bits better, at a cost of $\widetilde{O}(2^{ck})$, with c the exponent of finding the solutions in the combination routines.

2.4 CJS algorithm [7]

If we want each routine in the pipeline to be the same, we will have m routines, and we need $mr \simeq n$ in order to succeed. The total cost is then of $(k/p)^m$ queries, km qubits (excluding the quantum oracle overhead), a classical time in $\widetilde{O}(\sum_{i<m}(k/p)^i 2^k) = \widetilde{O}((k/p)^m 2^k)$ and a polynomial classical memory. If $k = \alpha\sqrt{n\log_2(n)}$, the query cost is in $L(1/(2\alpha)$, and the time cost is in $L(1/(2\alpha)+\alpha$. The classical cost is minimized for $\alpha = 1/\sqrt{2}$, which implies $k = \sqrt{n\log_2(n)/(2)}$, which leads to a quantum query cost in $L(1/\sqrt{2})$ and a classical time cost in $L(\sqrt{2})$.

Remark 1. The quantum query exponent of [7, Theorem 5.2] is not tight.

3 Subset-sum algorithms

It turns out that solving $x \cdot (\ell_1, \ldots, \ell_k) \mod 2^r = V$ with $x \in \{0,1\}^k$ for random instances can be done more efficiently than brute-force. Indeed, as we have a small number of solutions, it corresponds to a random instance of a subset-sum problem with a density close to 1, for which multiple algorithms have been proposed [15,2,11]. All these algorithms (classical and quantum) have an exponential complexity, in $\widetilde{O}(2^{cn})$, for a given constant $c < 1$. As we consider a polynomial-quantum memory algorithm, we will focus on the classical subset-sum algorithms.

These algorithms rely on list-merging techniques: the complete solution is contructed from lists of candidate partials solutions.

A simple example is the Schroppel-Shamir algorithm [15]. The space of possible solutions of $\sum_{i=1}^n \varepsilon_i x_i = V$ is split into 4 parts S_1, S_2, S_3, S_4, with $S_1 = \sum_{i \leq n/4} \varepsilon_i x_i$, and so on.

The lists contains the possible partial sum on a fourth of the variables. The intermediate lists S_{12} and S_{34} contains the partial sums on the first and second half of the variables.

Without any other technique, the splitting would not gain anything, as the two intermediate lists would be of size $2^{n/2}$. The Schroeppel-Shamir algorithm gains by guessing $S_{12} \mod 2^{n/4}$. With that guess, the list is expected to be of size $2^{n/4}$. Conversely, $S_{12} \mod 2^{n/4}$ imposes the value $V - S_{12} \mod 2^{n/4}$ for S_{34}. Hence all the lists are expected to be of size $\widetilde{O}(2^{n/4})$. The solution will only be found for the correct guess of the intermediate value, requiring $2^{n/4}$ guesses overall, for a total cost of $\widetilde{O}(2^{n/2})$ time, but only $\widetilde{O}(2^{n/4})$ memory.

Finally, the merging in itself is the efficient generation of the intermediate lists (and of the complete solution) from the previous lists. As we want to produce a list of values with a constrained sum, we can sort the first list, and then check for each element of the second list if there is an element leading to a correct sum in the first list. The cost of the merging is the cost of sorting the input list and constructing the output list, here $\widetilde{O}(2^{n/4})$.

The algorithm from [2] uses similar techniques, but with a different splitting. Instead of considering a subset of the variables, they considered a partial sum with a smaller number of terms in it. They also allowed the exponent of the subset sum in the intermediate step to lie in $\{-1, 0, 1\}$. With this approach, the merging has also to check for the consistency of the solutions, as the variables may overlap.

By splitting the sum in 8 and carefully choosing the size of the intermediate constraint and the ratio of -1, the authors of [2] obtained a complexity in $\widetilde{O}(2^{0.291n})$ in classical time and memory.

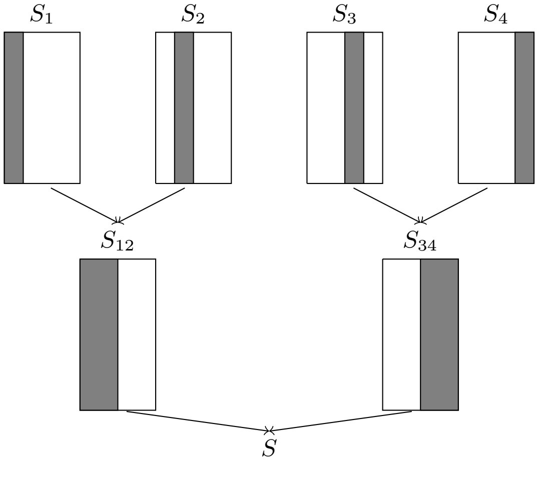

Fig. 1: Schroeppel-Shamir merging

It is also possible to devise polynomial-memory algorithms for this problem. The method in [2] consists in merging only two lists, using a memoryless collision-finding algorithm, with a cost in $\widetilde{O}(2^{0.72n})$.

In our case, we have some instances with more than one solution. As the expected number of solutions is fixed, this does not change the asymptotic cost for constructing all the solutions.

Finally, finding the pairs for Algorithm 2 consists in finding the solutions of the slightly different equation $\lfloor x \cdot (\ell_1, \ldots, \ell_k)/2^r \rfloor = V$. This equation is slightly different, but the exact same techniques can be applied, as the merging can also check for values lying in an interval. Hence, the cost is similar for both.

For the polynomial variant, we do not have an equality to check between the two lists. This can be solved by truncating the first $r+1$ bits of the partial sum. In that case, two partial sums that lead to the correct interval have more than 50% chance to have the same truncated value. In the case that we missed, we only need to redo this by adding 2^r to the partial sum before truncating. We will also have a fixed proportion of false positives that we need to discard. Overall, this does not change the complexity cost of $\widetilde{O}(2^{cn})$.

4 Improved hidden-shift algorithms

4.1 Improving the CJS algorithm

Instead of using brute-force to compute the output qubit value of each routine, we can use a subset-sum algorithm, which costs $\widetilde{O}(2^{cn})$. The only difference is that the classical part cost less. With $k = \alpha\sqrt{n \log_2(n)}$, we still have a quantum query cost in $L(1/(2\alpha))$, but the classical time cost becomes $L(1/(2\alpha) + c\alpha)$ and the optimal parameter becomes $k = \sqrt{n \log_2(n)/(2c)}$, leading to a quantum query complexity in $L(\sqrt{c/2})$ and a classical time complexity in $L(\sqrt{2c})$.

The classical time exponent of this hybrid algorithm is double the quantum query cost.

4.2 Minimizing the classical time

The classical cost of the previous approach is almost all in the first routine of the pipeline: the second routine is called k/p time less, the third $(k/p)^2$ time less, and so on. We can change the size of each routine and increase k to have each routine to cost overall roughly the same. The cost of a routine is in 2^{ck}, hence k can increase by $\log_2(n)/(2c)$ for each routine. The optimal point is for k increasing from 0 to $\sqrt{n \log_2(n)/c}$, by steps of $\log_2(n)/(2c)$. With these parameters, the routine i uses $k_i = i \log_2(n)/(2c)$. We need to have $\sum_{i<m} k_i \simeq n$, which implies $m \simeq 2\sqrt{cn/\log_2(n)}$. Hence, the cost of each routine will be in $2^{ck_m + o(k_m)}$. As $k_m \simeq \sqrt{n \log_2(n)/c}$, we obtain a quantum time cost in $L(\sqrt{c})$. The first routine does nothing, hence its cost is the number of queries, which is also in $L(\sqrt{c})$.

4.3 Enforcing a quadratic gap

Section 4.1 had an algorithm with $k_i = \beta\sqrt{n\log_2(n)}$, Section 4.2 used $k_i = i\alpha\sqrt{n\log_2(n)}$. We can generalize this and consider It turns out the two previous approaches can be generalized, and we can consider routine input sizes of the form $k_i = (i\alpha + \beta)\sqrt{n\log_2(n)}$, and for example ensure a quadratic gap between the number of queries and the classical time. If we want to have each m routine to cost the same, we still need $\alpha = \log_2(n)/(2c)$. This implies $m \simeq 2c\left(-\beta + \sqrt{\beta^2 + \frac{1}{c}}\right)\sqrt{n/\log_2(n)}$, hence $k_m \simeq \sqrt{\beta^2 + \frac{1}{c}}\sqrt{n\log_2(n)}$. The log of number of queries is in $c(k_m - k_1) = c\alpha m = c\left(-\beta + \sqrt{\beta^2 + \frac{1}{c}}\right)\sqrt{n\log_2(n)}$, as all the steps cost the same. Enforcing a quadratic gap between the two means that $\sqrt{\beta^2 + \frac{1}{c}} = 2\beta$, hence $\beta = 1/\sqrt{3c}$. With these values, the quantum query cost is in $L\left(\sqrt{\frac{c}{3}}\right)$, and the classical time cost is in $L\left(2\sqrt{\frac{c}{3}}\right)$.

As it is similar to the classical/quantum gap of Grover's algorithm, this approach can be interesting for cases where the targeted hardness is defined with respect to some exhaustive search, as for example in the security levels 1, 3 and 5 of the NIST call for post-quantum primitives.

4.4 Minimizing the number of queries

The simplest way to minimize to number of queries is to have only one routine, and directly use Algorithm 1 or Algorithm 2 to solve the whole problem: in that case, the number of queries is in $O(n)$ to obtain one target element, hence overall the quantum query cost is in $O(n^2)$. The classical cost is in $\widetilde{O}(2^{cn})$. Interestingly, this beats the asymptotic cost of the best known classical algorithm if $c < 1/2$.

4.5 Algorithms complexity

From Section 3, we can take $c = 0.291$ for a memory-heavy algorithm, or $c = 0.72$ for a polynomial-memory algorithm. This leads to the exponents of Table 1.

If we only consider quantum algorithms (or that classical and quantum time and memory are equivalent), then we have sightly different complexities. Currently, an algorithm with the exponent $c = 0.241$ has been proposed in [3], and one with exponent $c = 0.226$ in [11]. The costs are summarized in Table 2. It is to be noted that the non-polynomial quantum memory hidden shift algorithms we obtain perform asymptotically worse than the algorithms from [4,12,13], which use a different combination routine.

Using Grover's algorithm for a small number of queries produces an algorithm with the same cost as the exhaustive search, with a small number of queries. Overall, this is slightly worse than the approach of [8], which performs an exhaustive search, but achieves a linear number of queries.

Quantum query	Quantum time	Quantum memory	Subset-sum	Source
$L(0.5)$	$L(1)$	$O(n)$	Grover	Section 4.1
$L(1/\sqrt{2})$	$L(1/\sqrt{2})$	$O(n)$	Grover	Section 4.2
$O(n^2)$	$2^{0.5n}$	$O(n)$	Grover	Section 4.4
$L(0.283)$	$L(0.567)$	$L(0.283)$	[3]	Section 4.3
$L(0.491)$	$L(0.491)$	$L(0.491)$	[3]	Section 4.2
$O(n^2)$	$\widetilde{O}\left(2^{0.241n}\right)$	$\widetilde{O}\left(2^{0.241n}\right)$	[3]	Section 4.4
$L(0.274)$	$L(0.549)$	$L(0.274)$	[11]	Section 4.1
$L(0.475)$	$L(0.475)$	$L(0.475)$	[11]	Section 4.2
$O(n^2)$	$\widetilde{O}\left(2^{0.226n}\right)$	$\widetilde{O}\left(2^{0.226n}\right)$	[11]	Section 4.4

Table 2: Purely quantum algorithm costs

5 Conclusion

In this paper, we showed how to use subset-sum algorithms to reduce significantly the cost of a quantum hidden shift algorithm, and proposed different quantum/classical cost tradeoffs, allowing to divide the exponent by roughly 2.6 compared to [7], and even by 4.5 for the quantum query cost if we allow a quadratic gap between the classical time cost and the quantum query cost.

Improving the complexity. In order to obtain more efficient algorithms, one might study what happens if the combined elements can be chosen from a larger pool, as done in the quantum-memory heavy algorithms of [12]. This may allow to reduce the time cost at the expense of the quantum memory, while still offering a tradeoff between classical time and quantum query. Another approach would be to apply similar techniques to the algorithm of [13], if applicable.

References

1. Alagic, G., Russell, A.: Quantum-Secure Symmetric-Key Cryptography Based on Hidden Shifts. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT (3). LNCS, vol. 10212, pp. 65–93 (2017)
2. Becker, A., Coron, J., Joux, A.: Improved generic algorithms for hard knapsacks. In: Paterson, K.G. (ed.) Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6632, pp. 364–385. Springer (2011)
3. Bernstein, D.J., Jeffery, S., Lange, T., Meurer, A.: Quantum algorithms for the subset-sum problem. In: Gaborit, P. (ed.) Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7932, pp. 16–33. Springer (2013)

4. Bonnetain, X., Naya-Plasencia, M.: Hidden shift quantum cryptanalysis and implications. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11272, pp. 560–592. Springer (2018)
5. Bonnetain, X., Schrottenloher, A.: Quantum security analysis of CSIDH and ordinary isogeny-based schemes. IACR Cryptology ePrint Archive 2018, 537 (2018)
6. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: Csidh: An efficient post-quantum commutative group action. Cryptology ePrint Archive, Report 2018/383 (2018), https://eprint.iacr.org/2018/383
7. Childs, A.M., Jao, D., Soukharev, V.: Constructing elliptic curve isogenies in quantum subexponential time. J. Mathematical Cryptology 8(1), 1–29 (2014)
8. Ettinger, M., Høyer, P.: On Quantum Algorithms for Noncommutative Hidden Subgroups. In: STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings. LNCS, vol. 1563, pp. 478–487. Springer (1999)
9. Feo, L.D., Galbraith, S.D.: Seasign: Compact isogeny signatures from class group actions. IACR Cryptology ePrint Archive 2018, 824 (2018)
10. Feo, L.D., Kieffer, J., Smith, B.: Towards practical key exchange from ordinary isogeny graphs. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11274, pp. 365–394. Springer (2018)
11. Helm, A., May, A.: Subset sum quantumly in 1.17^{n}. In: Jeffery, S. (ed.) 13th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2018, July 16-18, 2018, Sydney, Australia. LIPIcs, vol. 111, pp. 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2018)
12. Kuperberg, G.: A Subexponential-Time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. SIAM J. Comput. 35(1), 170–188 (2005)
13. Kuperberg, G.: Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. In: Severini, S., Brandão, F.G.S.L. (eds.) 8th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2013, May 21-23, 2013, Guelph, Canada. LIPIcs, vol. 22, pp. 20–34. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013)
14. Regev, O.: A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space. CoRR (2004)
15. Schroeppel, R., Shamir, A.: 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)
16. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (1997)