IACR Communications in Cryptology
ISSN 3006-5496, Vol. 2, No. 1, 36 pages.

https://doi.org/10.62056/ayzojb0kr

A high-level comparison of state-of-the-art quantum algorithms for breaking asymmetric cryptography

Martin Ekerå[1,2] and Joel Gärtner[1,2]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
[2] Swedish NCSA, Swedish Armed Forces, Stockholm, Sweden

Abstract. We provide a high-level cost comparison between Regev's quantum algorithm with Ekerå–Gärtner's extensions on the one hand, and existing state-of-the-art quantum algorithms for factoring and computing discrete logarithms on the other. This when targeting cryptographically relevant problem instances, and when accounting for the space-saving optimizations of Ragavan and Vaikuntanathan that apply to Regev's algorithm, and optimizations such as windowing that apply to the existing algorithms. Our conclusion is that Regev's algorithm without the space-saving optimizations may achieve a per-run advantage, but not an overall advantage, if non-computational quantum memory is cheap. Regev's algorithm with the space-saving optimizations does not achieve an advantage, since it uses more computational memory, whilst also performing more work, per run and overall, compared to the existing state-of-the-art algorithms. As such, further optimizations are required for it to achieve an advantage for cryptographically relevant problem instances.

Keywords: Regev's algorithm · Cost estimates · Factoring · Discrete logarithms

1 Introduction

In August of 2023, Regev [Reg25][1] introduced a quantum factoring algorithm that may be perceived as a d-dimensional variation of Shor's factoring algorithm [Sho94, Sho97].

To factor an n-bit integer N, Regev raises the squares of the first $d = \lceil \sqrt{n} \rceil$ primes to short exponents. By using binary tree-based arithmetic, and square-and-multiply–based exponentiation, Regev achieves a circuit size reduction by a factor $\tilde{\Theta}(\sqrt{n})$ compared to the other state-of-the-art variations of Shor's algorithms that are in the literature. This reduction comes at the expense of having to perform $d + 4$ runs, however, and at the expense of using $O(n^{3/2})$ qubits of space in each run. Regev's algorithm furthermore relies on a heuristic number-theoretic assumption.

In October of 2023, Ragavan and Vaikuntanathan [RV23] reduced the space requirements to $\tilde{O}(n)$ qubits by using Fibonacci-based exponentiation. A few months later, in February of 2024, Ragavan and Vaikuntanathan [RV24][2] improved the constants in their analysis and assigned a new title to their pre-print. More recently, in April and May of 2024, Ragavan [Rag24] introduced further optimizations, and generalized the Fibonacci-based exponentiation so as to allow for tradeoffs between the circuit size and the space usage.

In November of 2023, Ekerå and Gärtner [EG24b][3] extended Regev's factoring algorithm to algorithms for computing discrete logarithms and orders quantumly in groups for

E-mail: ekera@kth.se (Martin Ekerå), jgartner@kth.se (Joel Gärtner)
[1]For the initial pre-print version of [Reg25], see ArXiv 2308.06572v1.
[2]For the pre-print version of [RV24], see ArXiv 2310.00899v2.
[3]For the initial pre-print version of [EG24b], see ArXiv 2311.05545v1.

Received: 2025-01-14 Accepted: 2025-03-11

which there exist a notion of *small* elements.[4] Furthermore, Ekerå and Gärtner explained how to factor integers completely via order finding. The resulting factoring algorithm is slightly more efficient per run compared to Regev's original algorithm. Note, however, that Ekerå and Gärtner's extensions rely on a slightly stronger heuristic number-theoretic assumption than Regev's original algorithm.

In April of 2024, Pilatte [Pil24] proved a version of this stronger assumption by using tools from analytic number theory. This allows Ekerå–Gärtner's extensions of Regev's algorithm to be used to unconditionally factor and compute discrete logarithms, although this requires selecting parameters in a manner that may not be preferable in practice.

1.1 Our contributions

Arguably, quantum algorithms for factoring integers and computing discrete logarithms are of interest primarily because such algorithms may be used to break currently widely deployed asymmetric cryptography — including but not limited to Rivest–Shamir–Adleman (RSA) [RSA78, BCR+19], Diffie–Hellman (DH) [DH76, BCR+18] and DSA [Nat94, Nat23], and their elliptic curve counterparts EC-DH [BCR+18] and EC-DSA [Nat23].

Limiting factors for when it will first become possible to execute a quantum algorithm is arguably the size and depth of the circuit for the algorithm, and the space usage of the circuit. Further limiting factors that are important to consider are the number of runs of the circuit that are required, and whether or not all of these runs have to be correct.

Although the circuit for Regev's algorithm is asymptotically smaller than the circuits for the other state-of-the-art variations of Shor's algorithms, the asymptotic advantage only kicks in for sufficiently large problem instances. Furthermore, the circuit for Regev's algorithm uses more space, and asymptotically more runs of the circuit are required to achieve a high success probability. Hence, it is not clear that Regev's algorithm has an advantage in practice for cryptographically relevant problem instances, and the same holds true for Ekerå–Gärtner's extensions.

To begin to resolve this situation, in this work, we provide a coarse high-level comparison between Regev's algorithm and Ekerå–Gärtner's extensions of it on the one hand, and some of the other state-of-the-art quantum algorithms [EH17, Eke20, Eke23, Eke21, Eke24b] on the other hand. In particular, we consider factoring RSA integers to break RSA, and computing discrete logarithms in subgroups to $\mathbb{Z}_N^*$ for N a large prime to break DH and DSA. We do not consider EC-DH and EC-DSA since these schemes work in elliptic curve groups that do not admit a trivial notion of small group elements.[5]

For factoring RSA integers and computing short discrete logarithms, we compare Ekerå–Gärtner's extensions of Regev's algorithm to Ekerå–Håstad's variation [EH17, Eke20, Eke23] of Shor's algorithm. For computing general discrete logarithms, we instead compare them to Ekerå's variation [Eke24b] of Shor's algorithm.

We account for Ragavan's generalization [Rag24] of the space-saving optimizations of Ragavan and Vaikuntanathan [RV24] that apply to Regev's algorithm and to Ekerå–Gärtner's extensions, and for optimizations such as windowing [GE21, Gid19, VMI05, VM08] that apply to most other variations of Shor's algorithms, including in particular to Ekerå–Håstad's and Ekerå's variations.

We consider a range of problem instance sizes from currently widely deployed to more conservative. Since all of the aforementioned schemes (RSA, DH and DSA) are being

[4] The extended algorithms are generic, and work in any group (since the required arithmetic can be implemented in the same way as for Shor's algorithms), but for the algorithms to achieve a practical advantage over Shor there must exist a notion of *small* elements in the group, as explained in [EG24b].

[5] It should be stated that a non-trivial notion was recently given in [BBP24] for certain special-form curves. Said work claims an asymptotic improvement, but as far as we can see it does not provide an improvement in practice for problem instances of cryptographically relevant sizes.

phased out, we do not foresee that larger key sizes than the sizes that we cost in this paper will be widely deployed in the future.

1.2 Methodology

Regev's algorithm [Reg25] and Ekerå–Gärtner's extensions [EG24b] use more space, and require more runs, than Ekerå–Håstad's [EH17, Eke20, Eke23] and Ekerå's [Eke24b] variations of Shor's algorithm — hereinafter referred to as the existing algorithms. This is true also when accounting for the space-saving optimizations of Ragavan and Vaikuntanathan [RV24], and Ragavan's generalization [Rag24] thereof. It follows that for Regev's algorithm and Ekerå–Gärtner's extensions to achieve an advantage per run in practice over the existing algorithms, they need to perform less work per run.

To compare the amount of work performed per run, we follow Ragavan and Vaikuntanathan [RV24, Rag24], and assume that we have a quantum circuit that maps

$$\left| \, u, v, t, 0^S \, \right\rangle \to \left| \, u, v, (t + uv) \bmod N, 0^S \, \right\rangle \tag{1}$$

for S the number of ancilla qubits, and for $u, v, t \in [0, N) \cap \mathbb{Z}$, where we furthermore require u and v to be invertible modulo N. All of the aforementioned quantum algorithms can be implemented by calling this circuit as a subroutine. Furthermore, for the parameterizations of the aforementioned algorithms that we consider, all other work performed is negligible compared to the calls to this circuit. Hence, we may roughly compare the amount of work that the algorithms perform — i.e. the circuit depth and size — by counting the number of calls that they make to this multiplication circuit. This allows us to ignore the low-level details of how the algorithms are implemented, which greatly simplifies the comparison.

For RSA, for N an RSA integer, we count the number of calls made to the multiplication circuit (1) in a run of Ekerå–Håstad's variation of Shor's algorithm, and of Regev's algorithm or of Ekerå–Gärtner's extension to factoring via order finding, respectively.

For DH and DSA, for N a large prime, we similarly count the number calls made to the multiplication circuit (1) in a run of Ekerå–Håstad's or of Ekerå's variation of Shor's algorithm, and of Ekerå–Gärtner's extension of Regev's algorithm, respectively.

Furthermore, we estimate the number of runs required to solve the problem instance with a given high success probability, and count the overall number of circuit calls required across all of these runs. This allows us to state both per-run and overall costs.

We consider N of bit length $n \in \{2048, 3072, 4096, 6144, 8192\}$. Our rationale for this choice is that $n \in \{2048, 3072, 4096\}$ are arguably the most common choices in current commercial cryptographic applications, whereas $n \in \{6144, 8192\}$ are more conservative choices. Note also that already for quite some time now, there has been a gradual shift away from RSA, DH and DSA — initially to EC-DH and EC-DSA, and more recently to post-quantum secure cryptography. Hence, we do not expect N of larger bit lengths to be widely deployed in the future. Rather, we expect the use of RSA, DH and DSA to continue to decrease, and eventually cease.

For DSA, that uses Schnorr groups, we consider subgroups of sizes selected based on the strength level estimates used by NIST, as given in [BCR+18, Tab. 25–26 in App. D on p. 133] and [NC23, Sect. 7.5 on p. 126]. For DH, we consider both safe-prime groups with short exponents and Schnorr groups. We select the logarithm length and subgroup size, respectively, based on the strength level estimates used by NIST. Note that for DH, the security typically depends on the short discrete logarithm problem in practice in protocols such as TLS [Gil16] and IKE [KK03].

1.3 Overview

In what follows, we first review Ekerå–Håstad's and Ekerå's variations of Shor's algorithm in Sect. 2. We then review Regev's algorithm and Ekerå–Gärtner's extensions of it in

Sect. 3. We describe the outcome of our cost comparison in Sect. 4, and summarize and conclude the paper in Sect. 5. We provide detailed tabulated results in App. A.

1.4 Notation

In what follows, we denote by log the base-two logarithm. We use $\lceil u \rceil$, $\lfloor u \rfloor$ and $\lfloor u \rceil$ to denote u rounded up, down and to the closest integer, respectively. We use the convention that empty products evaluate to one.

2 Existing variations of Shor's algorithms

In this section, we briefly review Ekerå–Håstad's [EH17, Eke20, Eke23] and Ekerå's [Eke24b] variations of Shor's algorithms [Sho94, Sho97], so as to introduce relevant notation, and so as to describe implementation details relevant to our cost comparison.

For detailed information on these algorithms, the reader is referred to the aforementioned references since our goal here is not to write a survey paper, nor to describe algorithms that are already in the literature. To facilitate reader comprehension, we do however start off by briefly introducing Shor's original algorithms [Sho94, Sho97] in Sect. 2.1 below, and then proceed to Ekerå–Håstad's and Ekerå's variations in Sect. 2.2 and Sect. 2.3, respectively.

Our rationale for considering these variations in particular is that they represent a part of the state of the art, and that they provide an advantage in our cost metric, which is sufficient to show that Regev's algorithm does not have the advantage.

2.1 Preliminaries

Let g be a generator of a finite cyclic group of order r, and let $x = g^d$ for some integer $d \in [0, r)$. The order-finding problem (OFP) is then the problem of finding r given g, whereas the discrete logarithm problem (DLP) is the problem of finding d given g, x, and optionally the order r of g. When $d \lll r$, the logarithm d and the DLP are said to be short, whereas the logarithm d and the DLP are said to be general when $d \in [0, r)$. For convenience, we write the group $\langle g \rangle$ multiplicatively.

Given a composite integer N, the integer factoring problem (IFP) is the problem of splitting N into a product of two non-trivial factors. The RSA IFP is the problem of splitting an RSA integer $N = pq$ into p, q — two large random distinct primes of equal bit lengths. When N is not of special form, the integer N and the IFP are said to be general.

2.1.1 Shor's original algorithms

At an overarching level, Shor's quantum algorithms [Sho94, Sho97] for the OFP and DLP may be said to induce the state

$$\frac{1}{\sqrt{L_a L_b}} \sum_{a=0}^{L_a-1} \sum_{b=0}^{L_b-1} \left| \, a, b, g^a x^{-b} \, \right\rangle \tag{2}$$

which — depending on how L_a, L_b are selected — is periodic in the logarithm d and/or in the order r. By applying quantum Fourier transforms (QFTs) to the first two control registers, reading out the resulting frequencies, and classically post-processing them, r and/or d may be probabilistically recovered — provided that L_a, L_b are sufficiently large.

The computation of $g^a x^{-b}$ to the last work register dominates the cost of the quantum circuit for inducing the above state. In practice this computation is typically implemented by pre-computing powers of $g, x^{-1}, g^2, x^{-2}, g^{2^2}, x^{-2^2}, \ldots$ classically, and then composing these quantumly conditioned on the qubits in the control register — i.e. via the square-and-multiply algorithm. This implies that the total exponent length, as given by $\log L_a + \log L_b$, essentially determines the cost of the quantum circuit when working in a fixed group $\langle g \rangle$.

Further details For the DLP, Shor originally proposed [Sho97, Sect. 6] to let $L_a = L_b = r$. More specifically, Shor assumes g to be a generator of $\mathbb{Z}_N^*$ for N a prime, which implies that g has known order $r = N - 1$. The order r is also used in the classical post-processing.

For the OFP, Shor originally proposed [Sho97, Sect. 5] to let $L_a > r^2$ be a power of two, and to let $L_b = 1$. More specifically, Shor assumes g to be a generator of a cyclic subgroup to $\mathbb{Z}_N^*$ for N an integer, and uses that $r < N$ to bound L_a.

For the IFP, Shor originally proposed [Sho97, p. 1498] to factor general integers N via a classical probabilistic reduction from the IFP to the OFP in $\mathbb{Z}_N^*$.

Shor [Sho97, Sects. 5–6] lower-bounds the probability of his algorithms succeeding in recovering the solution after a given number of runs of the quantum circuit. A fairly large number of runs are required to guarantee a high success probability via these bounds.

2.1.2 Subsequent improvements

Following Shor's groundbreaking publication [Sho94, Sho97] in 1994 several variations of Shor's algorithms have been introduced [Sei01, EH17, Eke20, Eke23, Eke21, Eke24b] that aim to reduce the total per-run exponent length. Amongst these, we find Ekerå–Håstad's algorithm [EH17, Eke20, Eke23] for the short DLP, that can efficiently compute d without knowledge of r when $d \lll r$. Ekerå–Håstad's algorithm can also be used to factor RSA integers, via a classical probabilistic reduction from the RSA IFP to the short DLP.

In 2001, Seifert [Sei01] proposed to make tradeoffs in Shor's algorithm for the OFP. In essence, Seifert's idea is to compute a small amount of information on r in each run of the quantum algorithm by selecting L_a only slightly larger than r, to perform many runs, and to then jointly post-process the resulting frequencies from all of these runs classically. Ekerå [Eke21, App. A] later proposed to use lattice-based post-processing for Seifert's algorithm, and estimated the number of runs required to guarantee a success probability $\geq 99\%$ for different choices of L_a — i.e. when making different tradeoffs.

Without initially being aware of Seifert's work, Ekerå and Håstad [EH17] proposed to make tradeoffs when introducing their algorithm for the short DLP in 2017. Ekerå subsequently extended the notion to a slightly modified variation [Eke24b] of Shor's algorithm for the DLP, and estimated the number of runs required to guarantee a success probability $\geq 99\%$ when making different tradeoffs in this algorithm [Eke24b], and in Ekerå–Håstad's algorithm for the short DLP [Eke20]. When not making tradeoffs, a single run of these algorithms suffices [Eke23, Eke24b] to guarantee a success probability $\geq 99\%$.

To the best of our knowledge, the aforementioned variations of Shor's algorithms represent the state of the art for solving the RSA IFP, short DLP and general DLP quantumly when excluding Regev's recent variation [Reg25] and its extensions [EG24b].

2.2 Ekerå–Håstad's variation of Shor's algorithm

Ekerå–Håstad's variation [EH17, Eke20, Eke23] of Shor's algorithm [Sho94, Sho97] (EHS) efficiently solves the short DLP quantumly in cyclic groups of unknown order. It furthermore efficiently breaks RSA via a reduction from the RSA IFP to the short DLP.

The algorithm induces the state

$$\frac{1}{\sqrt{2^{m+2\ell}}} \sum_{a\,=\,0}^{2^{m+\ell}-1} \sum_{b\,=\,0}^{2^{\ell}-1} \left| \, a, b, g^a x^{-b} \, \right\rangle \xrightarrow{\text{QFT}}$$

$$\frac{1}{2^{m+2\ell}} \sum_{a,\,j\,=\,0}^{2^{m+\ell}-1} \sum_{b,\,k\,=\,0}^{2^{\ell}-1} \exp\left(\frac{2\pi\mathrm{i}}{2^{m+\ell}}(aj + 2^m bk)\right) \left|\, j, k, g^a x^{-b} \,\right\rangle$$

where $g \in \mathbb{Z}_N^*$ in the cases we consider in this paper, for N an RSA integer or a prime, $x = g^d$ for d a short m-bit logarithm, and $\ell \sim m/s$ for s a tradeoff factor.

By increasing s, the amount of work that needs to be performed in each run of the algorithm is reduced, at the expense of more runs being required to solve for d in the classical post-processing. Each run yields approximately ℓ bits of information on d, so the idea is to perform $n \geq s$ runs and to jointly solve the n outputs for d. In practice, this is achieved by using classical lattice-based post-processing.

For an analysis of how many runs n are required for a given tradeoff factor s and logarithm length m when breaking RSA, or when solving short discrete logarithms in safe-prime groups, see [Eke20]. For an analysis of how much work is required when picking $\ell = m - \Delta$ for some small Δ and solving in a single run, see [Eke23].

2.2.1 Implementing the algorithm

Although there are more elaborate ways to implement EHS in an optimized fashion (see e.g. [GE21]), a straightforward way to implement the algorithm is to classically pre-compute powers of the group elements that are to be exponentiated, and to compose them quantumly under the group operation — in this case multiplication modulo N — conditioned on a control qubit. For this purpose, we essentially need a circuit that maps

$$\left|\, c_i, u \,\right\rangle \to \left|\, c_i, u \cdot v_i^{c_i} \bmod N \,\right\rangle$$

for v_i the classically pre-computed value, and $c_i \in \{0, 1\}$ the control qubit.

Such a circuit may be constructed from the circuit in (1) by loading $v_i^{c_i}$ — i.e. either 1 or v_i conditioned on c_i — into a quantum register, and then taking

$$\left|\, c_i, u, v_i^{c_i}, 0, 0^S \,\right\rangle \to \left|\, c_i, u, v_i^{c_i}, u \cdot v_i^{c_i} \bmod N, 0^S \,\right\rangle$$

after which we unload $v_i^{c_i}$, load $-v_i^{-c_i} \bmod N = -(v_i^{-1})^{c_i} \bmod N$, and take

$$\left|\, c_i, u, -v_i^{-c_i} \bmod N, u \cdot v_i^{c_i} \bmod N, 0^S \,\right\rangle \to \left|\, c_i, 0, -v_i^{-c_i} \bmod N, u \cdot v_i^{c_i} \bmod N, 0^S \,\right\rangle$$

after which we unload $-v_i^{-c_i} \bmod N$ to obtain

$$\left|\, c_i, u \cdot v_i^{c_i} \bmod N, 0, 0, 0^S \,\right\rangle$$

as desired. A few intermediary swaps are required. We may then proceed to process the next element and control qubit, recursively. As we run through $i \in [0, n_e) \cap \mathbb{Z}$, for n_e the total exponent length, we make a total of $2n_e$ calls to the circuit in (1).

Following the exponentiation, two QFTs are applied to the control registers, and the resulting frequency pair (j, k) read out. To save space in practical implementations, the two QFTs may be interleaved with the exponentiation, and the control qubit recycled [PP00, ME99]. A single qubit then suffices to implement the control registers for the exponentiation. Note also that small phase shifts may be dropped by using Coppersmith's [Cop02] approximate QFT.

As a further optimization, we may use windowing [GE21, Gid19, VMI05, VM08], and process the control qubits in a window of some given size w. That is to say, instead of loading $v_i^{c_i}$ and $-v_i^{-c_i} \bmod N$, we may use a quantum lookup table to load

$$\prod_{j\,=\,0}^{w-1} v_{i+j}^{c_{i+j}} \bmod N \qquad \text{and} \qquad -\prod_{j\,=\,0}^{w-1} v_{i+j}^{-c_{i+j}} \bmod N,$$

respectively, reducing the number of calls to the circuit in (1) to $2 \cdot \lceil n_e/w \rceil$. As explained above, the QFT may be interleaved with the exponentiation, and the control qubits recycled, in which case w qubits suffice to implement the control registers.

For each windowed multiplication, a table of 2^w values is first classically pre-computed, after which a quantum lookup into this table is performed. The classical pre-computation is not limiting, as we envisage quantum computations to be significantly more expensive than classical computations. However, the quantum lookup is not free. The cost of the lookup bounds how large w can reasonably be selected if we are to be able to neglect the lookup cost when comparing it to the cost of performing a large multiplication modulo N.

As explained in [GE21, Gid19], it is possible to go further and to combine windowing over the control qubits in the exponent with windowing over control qubits inside the multiplication circuit, and to optimize the multiplication circuit in various ways.

More specifically, in [GE21] for RSA-2048, these two levels of windowing are both of size 5, resulting in the algorithm performing lookups in a table of size 2^{10}. The window over the exponent decreases the number of multiplications by a factor of 5, and the window inside the multiplication circuit reduces the cost of this circuit by a factor of 5. Windowing hence essentially decreases the cost of the algorithm as a whole by a factor of 25.

For our cost estimates, we consider a black-box multiplication circuit, which we can not optimize, and we therefore can not use two levels of windowing. Instead, for simplicity, and for the same lookup cost, we use a single window of size $w = 10$ over the exponent to decrease the number of multiplications by a factor of 10. Note that this underestimates the efficiency gain from windowing in practice, as a larger gain would be possible if using a non-black-box multiplication circuit.

Note furthermore that selecting a larger window size could be beneficial for larger problem sizes. In practice, it would be reasonable to for example select $w = \Theta(\log \log N)$, but for simplicity we fix $w = 10$ for now in our cost comparisons. This when having a circuit for schoolbook multiplication in mind, as this seems to be the optimal choice [Gid19] for the problem sizes we consider in this paper.

2.3 Ekerå's variation of Shor's algorithm

Ekerå [Eke24b] has modified Shor's algorithm for computing general discrete logarithms so as to obtain an algorithm (ES) that efficiently solves the general DLP in cyclic groups of known order. The resulting algorithm induces the state

$$\frac{1}{\sqrt{2^{m+\varsigma+\ell}}} \sum_{a\,=\,0}^{2^{m+\varsigma}-1} \sum_{b\,=\,0}^{2^{\ell}-1} \left| \, a, b, g^a x^{-b} \, \right\rangle \xrightarrow{\text{QFT}}$$

$$\frac{1}{2^{m+\varsigma+\ell}} \sum_{a,\,j\,=\,0}^{2^{m+\varsigma}-1} \sum_{b,\,k\,=\,0}^{2^{\ell}-1} \exp\left(\frac{2\pi \mathrm{i}}{2^{m+\ell}} (aj + 2^{m+\ell-\varsigma} bk) \right) \left| \, j, k, g^a x^{-b} \, \right\rangle$$

where $g \in \mathbb{Z}_N^*$ in the cases we consider in this paper, for N a prime, $x = g^d$ for $d \in [0, r) \cap \mathbb{Z}$, m the bit length of the order r of g, and $\ell \sim m/s$ for s a tradeoff factor and ς a parameter to suppress noise. As for EHS, each run of ES yields approximately ℓ bits of information on d. The idea is to perform $n \geq s$ runs and to jointly post-process the outputs given r using lattice-based post-processing.

As is explained in [Eke24b], a problem that arises when making these modifications is that noise appears in the distribution. When solving in a single run with $s = 1$, this is not a problem since we may overcome the noise by searching a bit in the post-processing. When picking $s > 1$ and solving in $n \geq s$ runs, the parameter ς may be increased slightly above zero to suppress the noise. Hence, we pick $\varsigma = 0$ when solving in a single run and some small ς when solving in many runs.

Note that besides enabling tradeoffs, the modifications made to the algorithm enable qubit recycling by ensuring that the control qubits are separable. This is not the case in Shor's original algorithm where the control registers run over $[0, N-1) \cap \mathbb{Z}$. The modifications furthermore make the complexity of the algorithm depend on the bit length m of r, rather than on N as in Shor's original algorithm.

2.3.1 Implementing the algorithm

ES may be implemented in the same way as EHS, using control qubit recycling, approximate QFTs, windowing, etc. For further details, see Sect. 2.2.1.

The total exponent length is $n_e = m + \varsigma + \ell$, and there are $2\lceil n_e/w \rceil$ calls to the multiplication circuit, for w the window size.

3 Regev's algorithm and its extensions

In this section, we briefly review Regev's factoring algorithm [Reg25] and Ekerå–Gärtner's extensions [EG24b] thereof in Sects. 3.1–3.2 so as to introduce relevant notation, and so as to describe implementation details relevant to our cost comparison. In particular, we review the space-saving optimizations of Ragavan and Vaikuntanathan [RV24, Rag24].

For detailed information on these algorithms and optimizations, the reader is referred to the aforementioned references since our goal here is not to write a survey paper, nor to describe algorithms and optimizations that are already in the literature.

In Sects. 3.3–3.5, we furthermore provide a concrete analysis of how to parameterize the aforementioned algorithms so as to minimize the per-run cost of the quantum circuit as a function of the amount of work that the classical post-processing is allowed to perform. This concrete analysis is in itself a new contribution. It allows for significant cost savings, as illustrated in further detail in Sect. 4.

3.1 High-level overview

Regev's factoring algorithm [Reg25] and Ekerå–Gärtner's extensions [EG24b] thereof to computing discrete logarithms and orders, and for factoring via order finding, may be perceived as high-dimensional variations of Shor's algorithms [Sho94, Sho97]. More specifically, these quantum algorithms induce an approximation of a state proportional to

$$\sum_{\boldsymbol{z} \in \{-D/2,\ldots,D/2-1\}^{d+k}} \rho_R(\boldsymbol{z}) \left| z_1, \ldots, z_{d+k}, \prod_{i=1}^{d} a_i^{z_i + D/2} \prod_{i=1}^{k} x_i^{z_{d+i}+D/2} \bmod N \right\rangle \tag{3}$$

where $a_1, \ldots, a_d$ are small integers and $x_1, \ldots, x_k$ are arbitrary integers. Furthermore, $\boldsymbol{z} = (z_1, \ldots, z_{d+k})$, $\rho_R(\boldsymbol{z})$ is a Gaussian function of parameter R, and $D = 2^l$ is a power of two closely related to R. For Regev's original algorithm $k = 0$, whereas k is a small constant for Ekerå–Gärtner's extensions.

In analogy with Shor's algorithms, QFTs of the first $d + k$ control registers are then performed and the resulting frequencies read out. Provided that the parameters are appropriately selected, there will be a periodicity in the state that can be found, under a heuristic assumption, by classically post-processing the outputs from m independent runs of the circuit. This periodicity contains enough information to allow N to be efficiently factored classically, and for all discrete logarithm relations between the a_i and the x_i to be efficiently computed classically. As for Shor's algorithms, the per-run cost of the circuit is dominated by the cost of computing the product in the last work register.

The total exponent length of Regev's algorithm as given by $(d + k)l$ is essentially on par with that of Shor's algorithms when selecting parameters as Regev originally proposed.

However, Regev is able to leverage that $a_1, \ldots, a_d$ are small integers to compute the first part of the product using only $O(l)$ instead of $O(dl)$ large multiplications modulo N, as described in further detail in Sect. 3.2. For Ekerå–Gärtner's extensions, for which k is a small constant as stated above, the second part of the product can be computed in the same way as for Shor's algorithms, as described in further detail in Sect. 3.2.3, still resulting in $O(l)$ large multiplications modulo N.

There is an intricate relationship between value of d, the number of runs m, the value of l (that essentially determines the per-run cost) and the cost of the classical post-processing. Previous works have mainly considered the case $d \approx m \approx \sqrt{n}$ and $l = \Theta(\sqrt{n})$, but in Sects. 3.3–3.5 we provide a more advanced analysis that allows us to concretely select l as a function of d and m and the classical post-processing employed. Up to a certain limit, this allows us to increase d and m so as to decrease l and hence the per-run cost.

3.2 Implementing the algorithms

In this section, we first explain how to implement Regev's original algorithm for which $k = 0$, and then consider the case $k > 0$ as it is relevant to some of Ekerå and Gärtner's extensions.

3.2.1 Regev's original arithmetic

To compute the product over $a_1, \ldots, a_d$ in the last register in (3), Regev originally proposed to process the first d registers $z_1, \ldots, z_d$ bit by bit. Let

$$z_i + D/2 = \sum_{j=0}^{l-1} 2^j z_{i,j} \quad \text{for} \quad z_{i,j} \in \{0,1\}.$$

Then the product in the last register may be re-written as

$$\prod_{i=1}^{d} a_i^{z_i + D/2} \bmod N = \prod_{j=0}^{l-1} \left(\prod_{i=1}^{d} a_i^{z_{i,j}} \right)^{2^j} \bmod N = \prod_{j=0}^{l-1} c_j^{2^j} \bmod N$$

where we denote the inner product over i by c_j.

In Regev's algorithm, the a_i are small integers, so c_j is the product of small integers a_i raised to powers $z_{i,j} \in \{0,1\}$. For the values of d considered by Regev, the product c_j is furthermore guaranteed to be significantly smaller than N, ensuring that c_j can be computed quite efficiently by using a binary tree-based approach, as described by Regev.

To compute the full product given the c_j, Regev proposes to use the square-and-multiply algorithm: The first step is to compute c_{l-1} to an intermediate register. Next, the square modulo N of this register is computed to another intermediate register and multiplied with c_{l-2} modulo N. The algorithm continues this process recursively until c_0 is multiplied into the last intermediary register, which then contains the desired product. Finally, the result is copied out to a separate register, and the whole circuit run in reverse to uncompute the intermediary registers.

In total the algorithm thus performs $2(l-1)$ squarings modulo N, and $2(l-1)$ multiplications by c_j modulo N for different j. A squaring or multiplication modulo N can be performed in a single call to the circuit in (1). If the cost of computing and uncomputing the c_j is ignored, the algorithm can thus be implemented by calling said circuit $4(l-1)$ times. Unfortunately, however, the algorithm also requires $l + 1 = O(\log D) = O(\sqrt{n})$ registers, each of length n qubits, to hold the intermediate values generated by the squaring operations and the final result. This is necessary since squaring modulo N is not a reversible operation; it can not be efficiently performed in place.

Note also that windowing can not be applied to Regev's algorithm, since both operands passed to the multiplication circuit are quantum in his algorithm, whereas one of the operands is a conditionally loaded classical constant in most other variations of Shor's algorithms. Windowing is used to load the constant operand in an efficient manner: It critically relies on the values in the lookup table being classical constants that can be pre-computed. Only the lookup index is quantum.

3.2.2 Ragavan–Vaikuntanathan's space-saving optimizations

To circumvent the reversibility issue in Regev's original arithmetic, and the space increase to which it gives rise, Ragavan and Vaikuntanathan [RV23,RV24] have proposed to compute the product

$$\prod_{i=1}^{d} a_i^{z_i + D/2} \bmod N \tag{4}$$

in a different way, that does not require non-invertible squarings modulo N to be performed. They accomplish this feat by adapting ideas from Kaliski [KJ17] and decomposing the product with Fibonacci numbers in the exponent instead of powers of two, by writing

$$z_i + D/2 = \sum_{j=1}^{K} z_{i,j} F_j \qquad \text{for} \qquad z_{i,j} \in \{0, 1\},$$

where F_j is the j:th Fibonacci number, and K is the greatest integer such that $F_K \leq D$.

In a recent follow-up work, Ragavan [Rag24] generalizes this idea by defining generalized Fibonacci numbers via the recurrence

$$G_j^{(r)} = r G_{j-1}^{(r)} + G_{j-2}^{(r)}, \qquad G_1^{(r)} = G_0^{(r)} = 1,$$

for r some positive integer. Ragavan decomposes the product (4) with these generalized Fibonacci numbers in the exponent, by writing

$$z_i + D/2 = \sum_{j=1}^{K^{(r)}} z_{i,j} G_j^{(r)} \qquad \text{for} \qquad z_{i,j} \in \{0, 1, \ldots, r\},$$

where $K^{(r)}$ is the greatest integer such that $G_{K^{(r)}}^{(r)} \leq D$.

Note that the generalized Fibonacci numbers reduce to the ordinary Fibonacci numbers for $r = 1$, so we describe only Ragavan's generalization in what follows.

Note furthermore that re-writing a number in the generalized Fibonacci basis can be accomplished efficiently and essentially in-place, as explained by Ragavan [Rag24]. The cost is negligible compared to the cost of performing large multiplications modulo N, so we neglect it in our cost estimates.

The product (4) is rewritten as

$$\prod_{i=1}^{d} a_i^{z_i + D/2} \bmod N = \prod_{j=1}^{K^{(r)}} \left(\prod_{i=1}^{d} a_i^{z_{i,j}} \right)^{G_j^{(r)}} \bmod N = \prod_{j=1}^{K^{(r)}} c_j^{G_j^{(r)}} \bmod N \tag{5}$$

where, for each j, we denote the inner product over i by c_j.

As c_j is the product of d small integers a_i raised to powers $z_{i,j} \in \{0, 1, \ldots, r\}$, for relatively small d and r, it can be computed quite efficiently by generalizing Regev's binary tree-based arithmetic. For $d = \lfloor \sqrt{n} \rfloor$ and constant r as considered by Ragavan [Rag24], the cost of computing the c_j is negligible. However, in practice it is beneficial to select larger d and r, see Sect. 3.3, which may result in a non-negligible cost for computing the c_j. For technical reasons, Ragavan and Vaikuntanathan must also compute c_j^{-1}, and the cost of this computation may also be affected by selecting larger d and r. See Sect. 3.4 for details on how this cost is accounted for in our cost metric.

To compute the product (4), Ragavan introduces an additional parameter s that is a power of two such that r is a multiple of s. By using $\sim 2 \log s$ additional intermediary n-bit registers, Ragavan computes the product with $f(r, s) \cdot K^{(r)}$ large multiplications modulo N, where

$$f(r, s) = 2 \cdot (3r/s + 4 \log s + 7) \tag{6}$$

as described in [Rag24, Thm. 2.4]. Note that our expression for the cost differs from the expression in [Rag24] in that it depends directly on $K^{(r)}$ instead of estimating $K^{(r)}$ as $\log D / \log \beta$ for $\beta = (r + \sqrt{r^2 + 4})/2$. Note furthermore that we have doubled the cost compared to the expression in [Rag24], as the circuit described by Thm. 2.4 must be run twice — once to compute the product, and once in reverse to clean up intermediary values. Finally, note that this expression for the cost does not take into account the cost of computing the c_j, as this cost is negligible for the parameters considered by Ragavan.

3.2.3 Ekerå–Gärtner's extensions

As previously stated in the high-level overview, the quantum circuit for Ekerå–Gärtner's extensions are very similar to that of Regev's original algorithm: The circuits differ only in that not all of the elements included in the product in the last register of the state in (3) are small. Instead, in addition to the first d small elements $a_1, \ldots, a_d$, there are also k elements $x_1, \ldots, x_k$ that may be arbitrarily selected, for k a small constant.

The part of the product that depends on the small elements $a_1, \ldots, a_d$ can be computed in exactly the same way as for Regev's original algorithm [Reg25], with or without the space-saving optimizations of Ragavan and Vaikuntanathan [RV24, Rag24]. Hence, with $O(n^{3/2})$ qubits of memory, this part can be computed by calling the quantum circuit in (1) no more than $4 \log D$ times.[6] Alternatively, it can be computed by calling said circuit no more than $f(r, s) \cdot K^{(r)}$ times while using only $O(n)$ qubits of memory.

The part of the product that depends on the remaining k elements $x_1, \ldots, x_k$ can be computed using standard arithmetic in the same way as for the existing variations of Shor's algorithms, see Sect. 2.2.1. Hence, they can be computed by calling the multiplication circuit no more than $2k \log D$ times. Furthermore, windowing may be applied to compute this part of the product, again see Sect. 2.2.1. With a window size of w it is then sufficient to call the multiplication circuit $2 \lceil (k \log D)/w \rceil$ times.

The full product can thus be computed with $O(n^{3/2}) + S$ qubits of memory by calling the multiplication circuit $4 \log D + 2 \lceil (k \log D)/w \rceil$ times, or with $O(n) + S$ qubits of memory by calling the multiplication circuit $f(r, s) \cdot K^{(r)} + 2 \lceil (k \log D)/w \rceil$ times.

Ekerå–Gärtner's extensions of Regev's algorithms — hereinafter also referred to as EGR — furthermore differs from Regev's original algorithm in how the small elements $a_1, \ldots, a_d$ are selected: Whereas Regev selects them as the squares of d small primes, in EGR they are instead selected as d small primes. When comparing EGR to Regev's original algorithm, this somewhat decreases the cost of computing the first part of the product in the last work register in (3), as the bit lengths of $a_1, \ldots, a_d$ are halved.

Note that up until now, in the above descriptions of Regev's original algorithm and of EGR, the cost of computing the $c_j \le a_1^r \cdot \ldots \cdot a_d^r$ has been ignored, as it is typically negligible in our cost metric due to the c_j being less than N. This being said, for certain parameterizations that we consider, the c_j may be greater than N. For such parameterizations, we account for the cost of computing the c_j as described in Sect. 3.4. Furthermore, for such parameterizations, the choice of $a_1, \ldots, a_d$ as primes in EGR instead of as squares of primes reduces the cost of EGR in our metric.

[6] Ragavan [Rag24] discusses an option for reducing the cost from $4 \log D$ to $(2 + \epsilon) \log D$.

A further, more important, observation is that the fact that EGR uses smaller $a_1, \ldots, a_d$ than Regev's original algorithm allows larger values of d and r to be used with EGR without the cost of computing the c_j becoming non-negligible.

Finally, it should be stated that Ekerå and Gärtner [EG24b] make a slightly stronger heuristic assumption in their analysis compared to the assumption made by Regev [Reg25] in his original analysis. This being said, and as explained in detail by Ekerå and Gärtner [EG24b], there is no reason to doubt the validity of either of these heuristic assumptions. On the contrary, they can be seen to hold for special-form integers and small integers in simulations [EG24b, EG24a]. Furthermore, Pilatte [Pil24] has proved that a variant of these assumptions holds.

3.3 Selecting the constant C

As previously explained in Sect. 1.2, to compare the cost of EGR to EHS and ES, respectively, we count the number of calls to the circuit (1) for performing large multiplications modulo N that are required to solve a given problem instance, per run and overall.

For EGR, when using Regev's original arithmetic, the number of large multiplications modulo N that need to be performed per run depends on the value of $\log D \sim C\sqrt{n}$, and hence on the constant C. When using Ragavan's generalization of Ragavan and Vaikuntanathan's space-saving optimizations, it also depends on r and s via $f(r, s)$ and $K^{(r)} \approx \log D / \log \beta$ where $\beta = (r + \sqrt{r^2+4})/2$. In Ragavan and Vaikuntanathan's original work, $r = 1$, in which case β is the golden ratio $\phi = (1+\sqrt{5})/2$.

In [Reg25], Regev does not explore how to select the constant C as a function of the dimension d, number of runs m and quality of the lattice basis reduction. To explore how to select C, we must examine the heuristic assumption upon which his algorithm relies.

The heuristic assumption varies somewhat between Regev's original algorithm and Ekerå and Gärtner's extensions. In both cases, the assumption relates to a lattice $\mathcal{L}$ that is determined by the small integers $a_1, \ldots, a_d$ and the modulus N. Ekerå and Gärtner assume for their extensions that this lattice $\mathcal{L}$ has a basis that consists of vectors of length at most $T = \exp(O(n/d))$. This is also a sufficient assumption for Regev's original algorithm.

To analyze which value of C is sufficient, we let $T = \exp(\kappa n/d)$ for some constant κ. For Regev's variant of the heuristic assumption, with the specific choice of $d = \lceil \sqrt{n} \, \rceil$, it is natural to assume that selecting κ slightly larger than one is sufficient, as noted in [RV24]. It is also natural to assume that the somewhat stronger heuristic assumption that Ekerå and Gärtner rely on for their extensions holds for κ slightly larger than one.

In the primary analysis of his algorithm, Regev selects $d \approx \sqrt{n}$ and $m = d + 4$ when using LLL for the classical post-processing. However, Regev also notes that selecting a larger value for d may be beneficial when using a better — and typically computationally more expensive — lattice reduction algorithm for the classical post-processing, as it allows the per-run cost of the quantum circuit to be reduced. It can furthermore be seen that, even for a fixed value of d, increasing the number of runs m allows the algorithm to succeed with a smaller value of the constant C. As such, the choice of d and m has a large impact on the efficiency of the algorithm. Selecting $m \approx d \approx \sqrt{n}$, as previously primarily considered in [Reg25, RV24], is therefore probably not the best choice in practice.

For the classical post-processing to be successful in recovering the solution, Regev's analysis in [Reg25, Sect. 5] requires that

$$(2d+4)^{1/2} \cdot 2^{d+2} \cdot (d+5)^{1/2} \cdot T < \frac{\sqrt{2} \cdot 2^{C\sqrt{n}}}{6\sqrt{d}} \cdot (\det \mathcal{L})^{-1/m}$$

for the special case of $m = d + 4$. It can be seen that for arbitrary m this requirement corresponds to the requirement that

$$(m+d)^{1/2} \cdot 2^{(m+d)/2} \cdot (m+1)^{1/2} \cdot T < \frac{\sqrt{2} \cdot 2^{C\sqrt{n}}}{6\sqrt{d}} \cdot (\det \mathcal{L})^{-1/m}. \tag{7}$$

Inserting $T = 2^{\kappa n/d}$ and $\det \mathcal{L} < 2^n$ into (7) leads to the requirement

$$(m+d)^{1/2} \cdot 2^{\kappa n/d + (m+d)/2} \cdot (m+1)^{1/2} < \frac{\sqrt{2}}{6\sqrt{d}} 2^{C\sqrt{n} - n/m}$$

which, by taking logarithms on both sides, leads to the requirement

$$C > \frac{\kappa\sqrt{n}}{d} + \frac{m+d}{2\sqrt{n}} + \frac{\sqrt{n}}{m} + o(1)$$

where, as previously mentioned, the heuristic assumption should reasonably hold for $\kappa \approx 1$. If we let $\kappa = 1 + o(1)$, the above lower bound on C is minimized when $m \approx d \approx \sqrt{2n}$, which leads to the conclusion that $C > 2\sqrt{2} + o(1)$ is required.

However, we believe that in practice it is possible to select C that is significantly smaller than $2\sqrt{2}$. This is partly due to the fact that LLL [LLL82] performs significantly better in practice than Regev's analysis predicts, since it relies on the provable properties of LLL. Furthermore, it is due to the fact that the post-processing can use a computationally more expensive lattice reduction algorithm, such as BKZ [SE94], allowing an even smaller value of C to be used. This in turn reduces the per-run cost of the quantum circuit.

3.3.1 Using better lattice reduction

The factor $2^{(m+d)/2}$ in (7) is due to Regev's analysis relying on the provable properties of LLL. In particular, he relies on the Gram–Schmidt norm of two successive vectors of an LLL-reduced basis decreasing by at most a factor $\sqrt{2}$, giving rise to the factor $2^{(m+d)/2}$ for the full $(m+d)$-dimensional lattice.

Heuristic analyses of the performance of lattice reduction algorithms often rely on the so-called Geometric Series Assumption (GSA), under which the Gram–Schmidt norms of successive vectors decrease by some factor γ. Under the GSA, we can thus replace the factor $2^{(m+d)/2}$ in Regev's algorithm by a factor γ^{m+d}, with γ based on the heuristically estimated performance of the lattice reduction algorithm.

The performance of lattice reduction algorithms is not typically expressed directly in terms of the factor γ that is relevant for our analysis. Instead, the root-Hermite factor δ is often used, with the shortest vector $\boldsymbol{b}$ in a reduced basis of an n-dimensional lattice $\mathcal{L}$ having norm at most $\delta^n \cdot (\det \mathcal{L})^{1/n}$.

As the determinant of a lattice is equal to the product of the Gram–Schmidt norms of the vectors in any basis for the lattice, under the GSA we have that

$$\det \mathcal{L} = \|\boldsymbol{b}\|^n \prod_{i=0}^{n-1} \gamma^{-i} = \|\boldsymbol{b}\|^n \, \gamma^{-(n^2-n)/2},$$

and thus

$$\|\boldsymbol{b}\| = \gamma^{(n-1)/2} \cdot (\det \mathcal{L})^{1/n} \approx \gamma^{n/2} \cdot (\det \mathcal{L})^{1/n},$$

and we therefore heuristically take γ to be equal to δ^2. To estimate δ for BKZ [SE94] with varying block sizes, we use the formula of Chen [Che13] with special handling for small block sizes, in the same way as is done in the lattice estimator [APS15, ACD+24].

Using γ as determined by the heuristically estimated performance of the lattice reduction employed, the bound on C then becomes

$$C > \frac{\sqrt{n}}{d} + \log \gamma \left(\frac{m+d}{\sqrt{n}} \right) + \frac{\sqrt{n}}{m} + o(1) \tag{8}$$

with $\kappa = 1 + o(1)$. From this, we can see that the lower bound on C is minimized when $m \approx d \approx \sqrt{n/\log \gamma}$, leading to the requirement that $C > 4\sqrt{\log \gamma} + o(1)$.

3.4 On the cost of computing the c_j

Asymptotically, the cost of computing the c_j is negligible for as long as $a_1^r \cdot \ldots \cdot a_d^r \lll N$, since the multiplications that are required to compute this product then only involve comparatively small integers, and no reductions modulo N occur. When selecting $d = \lceil \sqrt{n} \, \rceil$ as Regev originally proposed, and $r = 1$, it is always the case that $a_1^r \cdot \ldots \cdot a_d^r \lll N$.

This being said, it may be beneficial to grow d and r so large that $a_1^r \cdot \ldots \cdot a_d^r > N$ for the purpose of suppressing C, as this may result in a smaller per-run cost, even if the cost of computing the c_j in each run is then no longer negligible.[7]

To estimate the cost of computing the c_j via Regev's binary tree-based approach in this setting, we count the number $M(d, r)$ of multiplications in the tree that yield a result $\geq 2^n$, and for which neither operand is equal to one, for n the bit length of N. This when taking the leaf nodes of the binary tree to be $(a_1^r, \ldots, a_d^r, 1, \ldots, 1)$, where the least number t of ones is padded on to the right for $d + t$ to become a power of two.

We note that other choices are possible: For instance, we could instead pad to the left, or balance the tree with the aim of having subtrees of approximately equal size at each level of the tree. For simplicity, we picked the above option of padding to the right since it is most in line with Regev's original proposal.

The cost of computing and uncomputing c_j and c_j^{-1} not captured by $f(r, s)$ is then $2^2 \cdot M(d, r)$ for each $j \in \{1, \ldots, K^{(r)}\}$. Recall furthermore that the product in (3) is first computed, copied out, and then uncomputed again, so the total cost is $2^3 \cdot M(d, r) \cdot K^{(r)}$.

Note that the above expression ignores the cost of all multiplications in the tree for which reductions modulo N do not occur. This is in accordance with our cost metric, but it should be stated that some of these multiplications still involve fairly large numbers. Note furthermore that growing d and r also has an additional cost in terms of increased space usage that is not captured by our cost metric.

3.5 Selecting parameters

To optimally select the parameters d, m, r and s for a given n, we need a method for estimating the least value of the constant C that guarantees a sufficiently high success probability in the classical post-processing. For this purpose, we use the bound (8) in Sect. 3.3 whilst ignoring the $o(1)$-term. Based on simulations, we believe that selecting C in this manner should lead to a success probability close to 1.

A reasonable objection to this approach is that the $o(1)$-term may potentially be significant for the problem instances we consider. However, we find that the least C yielded by the bound when ignoring the $o(1)$-term agrees quite well with C as estimated by the simulator in [EG24a]. Based on this observation, we conclude that the $o(1)$-term may be neglected for the problem instances we consider.

By using better lattice reduction, and selecting larger d and m, the constant C can be decreased, resulting in a lower cost for the quantum circuit. If one ignores the cost of the lattice reduction, and consider only the per-run cost, m can be selected arbitrarily large. This is in contrast to d which is still indirectly restricted, due to the fact that increasing d leads to an increased cost $M(d, r)$ of computing the c_j and c_j^{-1}, as detailed in Sect. 3.4.

To select r, d and s optimally given n, we exhaustively search through reasonable combinations of these parameters and select the combination that minimizes the cost in our

[7]We thank Seyoon Ragavan for raising this issue with us in private communication following the publication of the first version of this pre-print.

metric. As such, when factoring, we minimize $(f(r, s) + 2^3 \cdot M(d, r)) \cdot K^{(r)}$. When computing discrete logarithms, we instead minimize $(f(r, s) + 2^3 \cdot M(d, r)) \cdot K^{(r)} + 2 \lceil (\log D)/w \rceil$ since we then use a variant [EG24b, Sect. 3.1] of EGR for which $k = 1$.

4 Cost comparisons and results

In App. A, we present several cost comparisons between the existing variations of Shor's algorithms on the one hand, and EGR with the space-saving optimizations of Ragavan and Vaikuntanathan on the other. In particular, we focus on cryptographically relevant instances of the RSA integer factoring problem (IFP) and the discrete logarithm problem (DLP).

The most interesting comparison is between Regev's algorithm, with all currently available improvements and optimizations applied, and somewhat optimized versions of the existing variations of Shor's algorithms. Some details differ between the IFP and DLP, and between the different problem instances for the DLP that we consider, whilst other overarching aspects remain the same.

In particular, Ragavan's generalization [Rag24] of the aforementioned space-saving optimizations yields the lowest cost in our metric. Furthermore, to optimize the cost of Regev's algorithm in our metric, the best lattice reduction algorithm that it is practically feasible to use should be employed for the post-processing. Given an estimate of the performance of this lattice reduction algorithm, the optimal cost in our metric is achieved by selecting d, m, r and s as described in Sect. 3.5.

In practice, the best lattice reduction is achieved by the BKZ algorithm [SE94] with as large a block size as it is practically feasible to select. It is not clear exactly what the limit is for the block size. So as to avoid overestimating the cost of Regev's algorithm, given that we intentionally seek to bias our comparisons to be in its favor, we select a block size of 200 in our primary comparisons. This is clearly a larger block size than what is practically feasible to use today, yet not too great of an exaggeration. We heuristically estimate the performance of BKZ, and select the minimum C with precision 0.01 that allows the post-processing to succeed, again see Sect. 3.5.

In addition to our primary comparisons that use BKZ-200, we provide a set of additional comparisons for RSA to illustrate the benefits of the different optimizations that we apply to Regev's algorithm, and of windowing. Some of these comparisons use LLL.

Finally, it should be noted that our cost metric cannot be used to make useful comparisons between Regev's original algorithm without Ragavan–Vaikuntanathan's space-saving optimizations and the pre-Regev variations of Shor's algorithm. In Sect. 4.3, we detail why this is the case, and why a much more advanced cost metric would be needed to fairly make such comparisons.

4.1 Comparisons for RSA

In this section, we compare Regev's algorithm that solves the IFP to EHS that solves the RSA IFP. In particular, we focus on Ekerå–Gärtner's extension of Regev's algorithm to factoring via order finding [EG24b, App. A] since it allows for slightly better flexibility in how parameters are selected. In turn, this allows the algorithm to achieve a slightly better efficiency in our cost metric.

We consider n-bit RSA integers N for $n \in \{2048, 3072, 4096, 6144, 8192\}$ so as to model RSA-2048 up to and including RSA-8192. By RSA integer we mean an integer with two random prime factors of identical bit lengths.

For EHS, we consider both the case where we do not make tradeoffs and solve in a single run, and the case where we make tradeoffs with a reasonably large tradeoff factor. For further details, see App. A.

To start off, we first provide a baseline comparison in Sect. 4.1.1 between Regev's algorithm parameterized as originally proposed by Regev, and EHS without windowing. For all other comparisons in Sects. 4.1.2–4.1.6, we compare EGR to EHS unless otherwise stated, and apply windowing with a window size of $w = 10$ to EHS.

The comparisons in Sects. 4.1.2–4.1.6 differ only with respect to how Regev's algorithm and its extensions are parameterized. We select optimal d and m as described in Sect. 3.5, and explore different choices of lattice reduction algorithms, and of using the generalized Fibonacci-based exponentiation proposed by Ragavan.

For an overview of how the different parameterizations compare to EHS, see Fig. 1 where the advantage of EHS over EGR is plotted as a function of the problem size for the different parameterizations considered in Sects. 4.1.2–4.1.5. The figure is based on the data tabulated in Tabs. 2–5 in App. A.

4.1.1 A basic baseline comparison

In App. A.1.1, for Regev's algorithm, we use LLL, and let $d = \lceil \sqrt{n} \rceil$ and $m = d + 4$, as originally proposed by Regev. Furthermore, we use Ragavan's generalization of the space-saving optimizations with $r = 1$ leading to $20K$ operations in the form of large multiplications modulo N being required per run.

As may be seen in Tab. 1 in App. A.1.1, Regev's algorithm performs approximately between a factor two to four fewer operations per run than EHS when not making tradeoffs. However, EHS requires significantly fewer operations overall when not making tradeoffs, as it then only requires a single run. Hence, if the goal is to optimize the overall work, not making tradeoffs is typically preferable.

By making tradeoffs, EHS has an advantage over Regev's algorithm, both per run and overall, for RSA-2048. For RSA-3072, the per-run cost is essentially on par, but EHS has the overall advantage. For RSA-4096 up to and including RSA-8192, EHS retains the overall advantage, but it has a per-run disadvantage.

Asymptotically, Regev's algorithm has the per-run advantage, and this is consistent with the behavior we observe in Tab. 1. This is the case for all the parameterization of Regev's algorithm that we consider for the RSA IFP, but the break-even point differs between the parameterizations.

The per-run cost for Regev's algorithm is essentially determined by the value of the constant C, see Sect. 3.3. As may be seen in Tab. 1, and as previously pointed out in [EG24b], it suffices to select $C \approx 2$ when the algorithm is parameterized as in this baseline comparison. By selecting a better parameterization, and potentially also using a better lattice reduction algorithm, the value of C can be decreased.

4.1.2 Using LLL and $r = 1$

In App. A.1.2, for EGR, we use LLL, select $r = 1$, and select optimal d and m. Furthermore, we apply windowing with $w = 10$ for EHS from this point on.

As may be seen in Tab. 2 in App. A.1.2, selecting $C \approx 1$ for EGR is sufficient with this parameterization. The cost per run for EGR with this parameterization is thus approximately halved compared to the baseline for which $C \approx 2$. At the same time, since we select larger m with this parameterization, the overall cost for EGR increases by approximately a factor of two compared to the baseline, leading to EHS having a significant overall advantage — but our goal in this comparison is to optimize for the per-run cost.

The above having been said, even though the per-run cost of EGR decreases by a factor of two compared to the baseline, the per-run gap to EHS in fact increases, thanks to windowing having been applied. Windowing reduces both the per-run cost and the overall cost of EHS by approximately a factor of $w = 10$. EHS therefore now has the per-run advantage for RSA-2048 up to and including RSA-8192. This underscores how important an optimization windowing is in practice.

From this point on, we apply no further optimizations to EHS, so the costs in Tab. 2 for EHS are the costs that EGR must go below to achieve an advantage.

4.1.3 Using BKZ-200 and $r = 1$

In App. A.1.3, we replace LLL with BKZ-200, select $r = 1$, and optimal d and m.

As may be seen in Tab. 3 in App. A.1.3, with this parameterization, it is sufficient to select $C \approx 0.5$. Thus, if BKZ-200 can be used for post-processing instead of LLL, the per-run cost of the quantum algorithm can be decreased by approximately a factor of two. However, this reduction comes at the cost of approximately doubling the number of runs required, so the overall cost of the quantum algorithm is approximately the same.

By using BKZ-200 in the post-processing, EGR now again outperforms EHS per run when not making tradeoffs, for RSA-4096 up to RSA-8192. For RSA-4096, the two algorithms are essentially on par. Note however that, for all problem instances, including the aforementioned instances, the overall cost of EGR is more than 400 times greater than that of EHS when not making tradeoffs.

By making tradeoffs, EHS outperforms EGR, both per run and overall, for RSA-2048 up to and including RSA-8192. For EGR to achieve a per-run advantage compared to EHS when making tradeoffs, even for the largest problem instances we consider, its cost must decrease by almost a factor of two. The first multiple of 1024 bits for which EGR achieves a per-run advantage with this parameterization is for RSA-27648, which has a classical strength level of 336 bits in the NIST model [NC23, Sect. 7.5 on p. 126].

4.1.4 Using LLL and optimal r

In App. A.1.4, we once again use LLL, select optimal r and s, and optimal d and m.

As may be seen in Tab. 4 in App. A.1.4, the effect of picking optimal r and s in Ragavan's generalization of the space-saving optimizations has a similar effect on the per-run cost to that of using BKZ-200 instead of LLL, and selecting $r = 1$ and optimal d and m, see Tab. 3 in App. A.1.3 and compare. Meanwhile, the overall cost is decreased by almost a factor of two compared to the overall cost in Tab. 3.

EGR again outperforms EHS per run when not making tradeoffs, for RSA-6144 and RSA-8192, and is on par with EHS for RSA-4096. However, its overall cost is at least 200 times greater than that of EHS. When making tradeoffs, EHS outperforms EGR, both per run and overall, for RSA-2048 up to and including RSA-8192.

Note than an important distinction between the parameterization in this section and the one in and Sect. 4.1.3 is that the post-processing in this section is feasible to execute in practice, whereas the post-processing in Sect. 4.1.3 is impractical.

4.1.5 Using BKZ-200 and optimal r

In App. A.1.5, we use BKZ-200, select optimal r and s, and optimal d and m.

As may be seen in Tab. 5 in App. A.1.5, by using all available optimizations, and BKZ-200 for the post-processing, we achieve a significantly lower per-run cost than for any of the parameterizations previously considered. Compared to using LLL with optimal r, the per-run cost decreases by slightly more than a factor of $4/3$. The improvement achieved by using better lattice reduction is thus smaller in this case where we use optimal r compared to the case when $r = 1$, see Sect. 4.1.3.

Note that for RSA-3072, selecting optimal d and r results in a non-negligible cost $M(d, r)$ of computing the c_j. For all other problem instances sizes considered, this cost is negligible.

This parameterization of EGR achieves a per-run advantage over EHS, when not making tradeoffs, for RSA-3072 up to RSA-8192. Furthermore, for RSA-8192, it only has a slight per-run disadvantage to EHS when making tradeoffs. Note, however, that EHS still has a large overall advantage for RSA-2048 up to and including RSA-8192.

The first multiple of 1024 bits for which EGR achieves a per-run advantage with this parameterization is for RSA-13312, which has a classical strength level of 248 bits in the NIST model [NC23, Sect. 7.5 on p. 126].

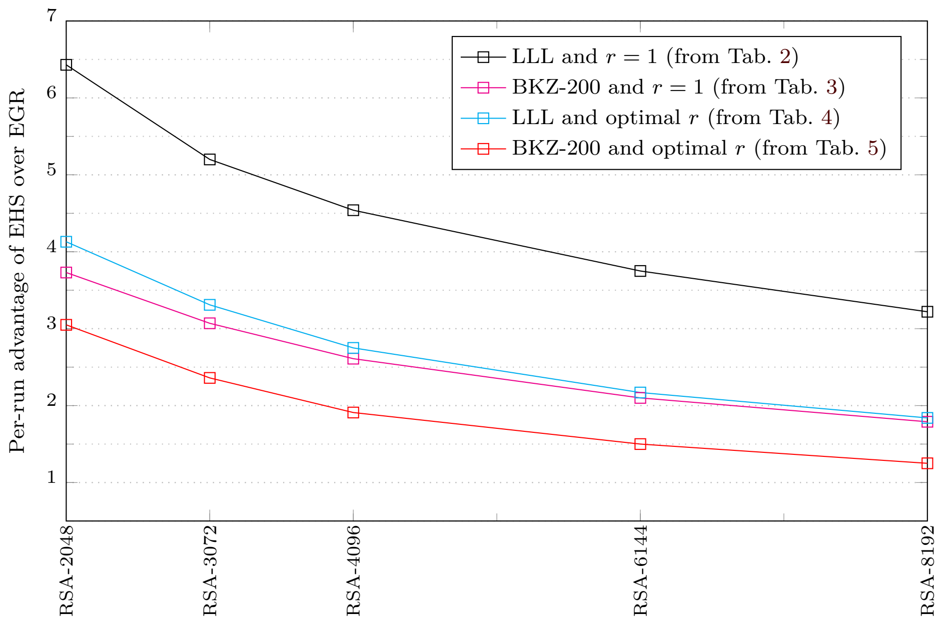

Figure 1: A plot of the per-run advantage of EHS over EGR for the RSA IFP for the parameterizations of EGR considered in Tabs. 2–5 in Apps. A.1.2–A.1.5. See also Sects. 4.1.2–4.1.5 for details on the parameterizations and tables.

The per-run advantage of EHS over EGR is plotted in Fig. 1 for the parameterization considered in this section, and for the parameterizations considered in Sects. 4.1.2–4.1.4, so as to provide an overview and to facilitate comparisons. As may be seen in Fig. 1, it is only when using BKZ-200, selecting optimal r and s, and optimal d and m, as in this section, that the per-run cost of EGR can be brought to be close to that of EHS for RSA-8192. Asymptotically, EGR has the advantage for all four parameterizations, but the asymptotic advantage only kicks in for sufficiently large problem instances.

4.1.6 Using perfect reduction and optimal r

In App. A.1.6, we use perfect lattice reduction, as modelled by letting $\gamma \to 1$ (in the analysis in Sect. 3.3.1), select optimal r and s, and optimal d and m. Note that this implies that $m \to \infty$, leaving the bound on C as $C > \sqrt{n}/d$.

Needless to say, this is not a realistic cost comparison since it is completely impractical to use perfect lattice reduction. Rather, its use essentially corresponds to the situation in the curious corollary by Regev in the introduction of [Reg25], which states that a quantum circuit for factoring with essentially linear size in n exists if lattice problems are easy to solve classically. In our model, this is captured by C not being constant. Rather, increasingly smaller values of C can be used as the problem instance size grows larger.

In App. A.1.6, we furthermore compare both Regev's original algorithm and EGR to EHS. In EGR, the small elements $a_1, \ldots, a_d$ are the first d primes, whereas in Regev's original algorithm, they are the squares of the first d primes. This implies that the cost of computing the c_j is somewhat higher for Regev's original algorithm than for EGR.

When selecting optimal parameters under our cost metric, this leads to smaller values of d and r being selected for Regev's original algorithm than for EGR since the cost $M(d, r)$ is higher for Regev's original algorithm than for EGR. The better the lattice reduction, the greater the impact of this difference between the algorithms. There is hence a significant performance difference between Regev's original algorithm and EGR when using perfect lattice reduction. This may be seen in Tab. 6 in App. A.1.6, where for each problem instance size, the top line is for Regev's original algorithm and the bottom line for EGR.

Even in the extremely biased comparison in Tab. 6, EHS does however retain a per-run advantage over Regev's original algorithm for RSA-2048 up to and including RSA-4096. Meanwhile, EHS retains a per-run advantage over EGR for RSA-2048 and RSA-3072. For larger problem instance sizes, the per-run cost of EHS is somewhat higher than that of Regev's original algorithm and EGR, respectively.

Note that we do not provide a comparison of the overall costs of the algorithms in Tab. 6. This is because the overall number of operations tends to infinity with m for Regev's algorithm and EGR, giving EHS an infinite overall advantage.

4.2 Comparisons for discrete logarithms

In addition to the above comparison for the RSA IFP, we provide analogous comparisons for the DLP in r-order subgroups to $\mathbb{Z}_N^*$, for $N = 2ur + 1$ a large n-bit prime, and r a prime. We use the model of NIST, as given in [BCR+18, Tab. 25–26 in App. D on p. 133] and [NC23, Sect. 7.5 on p. 126], to estimate the classical strength level z of N. As for RSA, we consider $n \in \{2048, 3072, 4096, 6144, 8192\}$. We consider both safe-prime groups for which $u = 1$, and Schnorr groups for which r is of length $2z$ bits. Both of these groups have a classical strength level of z bits.

In the case of safe-prime groups, we consider the short DLP where the logarithm is of length $2z$ bits, and the general DLP where the logarithm is on $[0, r)$. Again, both of these problems have a classical strength level of z bits. Note that cryptographic applications are typically based on the short DLP since it is much more efficient, but we nevertheless also include the general DLP in our comparison.

When using EGR to compute discrete logarithms, the algorithm cannot directly leverage that the r-order subgroup is small, as is the case for Schnorr groups, nor that the logarithm is short, so its cost is the same for all three parameterizations. We compare EGR to ES for the general DLP in Schnorr groups and safe-prime groups, and to EHS for the short DLP in safe-prime groups.

Note that in [EG24b], several extensions of Regev's algorithm are given for the DLP that use $k \in \{1, 2\}$, for k as defined in Sect. 3. In our comparison, for EGR, we use an extension with $k = 1$, necessitating either doubling the number of runs or performing some pre-computation for the specific choice of generator. We do not include these costs in the cost we report for EGR. The high-level outcome of the comparison would not be affected by choosing an extension with $k = 2$, but we select one with $k = 1$ to minimize the cost in our metric.

4.2.1 General DLP in safe-prime groups

As may be seen in Tab. 7 in App. A.2.1, EGR achieves a slight per-run advantage for 6144-bit and larger moduli, when comparing against ES and making tradeoffs. As for the overall cost, ES has the advantage by more than a factor 220 when not making tradeoffs, for all of the problem instance sizes considered.

Note that the reason for why EGR achieves an advantage for the general DLP, but not for the RSA IFP, is not because of differences between Ekerå–Gärtner's extensions of

Regev's algorithm to the IFP via the OFP, and to the DLP, respectively. Rather, it is because EHS for the RSA IFP achieves a reduction in the total control register length by using an efficient reduction from the RSA IFP to the short DLP.

Note furthermore that, as stated above, the cryptographically relevant cases are short discrete logarithms in safe-prime groups, and discrete logarithms in Schnorr groups, respectively, see the next two sections.

4.2.2 Short DLP in safe-prime groups

As may be seen in Tab. 8 in App. A.2.2, when the logarithm is short, EHS has a considerable advantage, both per run and overall, for 2048-bit moduli up to and including 8192-bit moduli. This is because EHS is specifically tailored to leverage the fact that the logarithm is short, whereas the complexity of EGR depends on the size of $\mathbb{Z}_N^*$. It is hence the same irrespective of whether the logarithm is short or full length. A strongly contributing reason for the advantage of EHS over EGR is the use of windowing, but even without windowing EHS would have slightly lower per-run cost than EGR, and also a significant overall advantage over EGR.

4.2.3 DLP in Schnorr groups

As may be seen in Tab. 9 in App. A.2.3, for Schnorr groups, ES has a considerable advantage, both per run and overall, for 2048-bit moduli up to and including 8192-bit moduli. Again, this is because ES is specifically tailored to leverage the fact that the subgroup is small, whereas the cost of EGR depends on the size of $\mathbb{Z}_N^*$. The cost of EGR is hence the same irrespective of whether g generates $\mathbb{Z}_N^*$, or a small subgroup of $\mathbb{Z}_N^*$. A strongly contributing reason for the advantage of ES over EGR is the use of windowing, but even without windowing ES would have slightly lower per-run cost than EGR, and also a significant overall advantage over EGR.

4.2.4 On the asymptotic advantage of Ekerå–Gärtner

In analogy to the situation for the RSA IFP, EGR achieves an asymptotic advantage over ES for the general DLP. This may be seen in Tab. 7 in App. A.2.1 where the advantage of ES decreases with the size of the problem instance.

For the short DLP, and the DLP in Schnorr groups, the situation is quite different, however. In these cases, the total exponent length in EHS and ES is a small multiple of the strength level $z = O(n^{1/3} \log^{2/3} n)$, see the NIST model [NC23, Sect. 7.5 on p. 126]. The number of large multiplication modulo N that need to be performed is twice the total exponent length. Meanwhile, $\Theta(n^{1/2})$ such operations need to be performed in EGR.

Note that in Tab. 8–9 in App. A.2.2–A.2.3, the advantage of EHS and ES does decrease with the size of the problem instance, but this phenomenon is not indicative of the asymptotic behavior.

4.3 More refined cost metrics

Our cost metric is simplistic, yet sufficient to enable us to compare the algorithms we consider in this work, and to draw several important conclusions.

In particular, our comparisons indicate that making a more detailed physical cost estimate of Regev's algorithm with Ragavan–Vaikuntanathan's space-saving optimizations is premature: Although such a cost estimate would give a clearer picture of the concrete cost of the algorithm, our much simpler analysis already shows that previous variations of Shor's algorithms would outperform it for cryptographically relevant problem instances. Hence, it would be fruitful to attempt to find further optimizations of the algorithm before proceeding to make physical cost estimates.

A limitation to our simplistic cost metric is that it does not account for the memory usage of the algorithms we compare. However, even with Ragavan–Vaikuntanathan's space-saving optimizations, Regev's algorithm still uses more memory than the variations of Shor's algorithm to which we compare it. As such, by not accounting for the memory usage, our metric biases the comparison in favor of Regev's algorithm. Since our conclusion is that Regev's algorithm does not have the advantage, this bias is not an issue.

To fairly compare Regev's original algorithm without Ragavan–Vaikuntanathan's space-saving optimizations to the previous variations of Shor's algorithms would require us to somehow account for the memory usage of the algorithms: As Regev's original algorithm uses a lot more memory than the previous variations, using our simplistic cost metric that ignores memory costs would heavily bias the comparison in favor of Regev's algorithm. At the same time, the metric would often indicate that Regev's algorithm has the advantage, and hence it would not be possible to draw any useful conclusions from the comparison.

As we see it, it is very much non-trivial to define a simplistic cost metric that fairly accounts for both the computations and the memory usage of the algorithms that we are interested in comparing. In particular, defining such a cost metric would require us to make a range of assumptions regarding the relative costs of computations and memory usage. A further complicating factor is that a much more detailed exploration of potential optimizations and memory tradeoffs for the algorithms would need to be undertaken to enable a fair comparison. Yet another complicating factor is that Regev's original algorithm uses both computational and non-computational memory[8], and that non-computational memory may be less expensive to implement in practice than computational memory.

From this, we conclude that a full-blown physical cost estimate would in essence be required to fairly compare Regev's original algorithm to the previous variations of Shor's algorithms. Developing such a physical cost estimate is far beyond the scope of this paper.

5 Summary and conclusion

Regev's factoring algorithm [Reg25] with the space-saving optimizations of Ragavan and Vaikuntanathan [RV24], as generalized by Ragavan [Rag24], does not achieve an advantage over the existing state-of-the-art variations of Shor's algorithms for cryptographically relevant problem instances. Further optimizations are required for it to achieve such an advantage. The same holds true for Ekerå–Gärtner's extensions to computing discrete logarithms, and to factoring via order finding.

This when using the number of large multiplications modulo N per run as the cost metric in a black-box model, and ignoring the fact that the space usage, and the number of runs required, is larger for Regev's algorithm and Ekerå–Gärtner's extensions. If we would instead have used a different cost metric that also accounts for the space usage, such as the spacetime volume, the difference in the advantage would have been even more pronounced.

It would also have been more pronounced if we had attempted to account for the overhead of quantum error correction. The fact that Regev's algorithm and Ekerå–Gärtner's extensions typically require many more runs than the existing state-of-the-art algorithms drives up the cost of the error correction. This is because the error correction has to be parameterized so as to ensure that all runs are correct with sufficiently high probability. Alternatively, the erroneous runs can be filtered out before the post-processing, or the erroneous runs jointly post-processed alongside the correct runs, as proposed by Ragavan and Vaikuntanathan [RV24], and by Ekerå and Gärtner [EG24b], respectively. However, this requires increasing C, driving up the per-run cost of the algorithm in our metric, and it requires making even more runs.

[8]Memory to which we can swap out a state for an extended period of time and then swap it back in at a later time when it is needed for the uncomputation in Regev's arithmetic.

Treating the multiplication circuit as a black box precludes the use of known optimizations that would benefit the existing state-of-the-art variations of Shor's algorithm, but that are not directly applicable to Regev's algorithm and Ekerå–Gärtner's extensions, further biasing the comparison in our cost metric. In practice, choosing the multiplication circuit is non-trivial. The same circuit need not necessarily be optimal for all of the aforementioned algorithms, depending on what other optimizations can be applied, but going into details on the implementation of the multiplication circuit is beyond the scope of this work.

In summary, the cost metric we use in this work is deliberately biased in favor of Regev's algorithm and Ekerå–Gärtner's extensions. This is not an issue, however, when our comparison shows that the existing state-of-the-art variations of Shor's algorithm have the advantage. We can then still draw valid conclusions.

Regev's algorithm [Reg25] without the space-saving optimizations does achieve an advantage over Ekerå–Håstad's algorithm for cryptographically relevant instances of the RSA IFP.[9] This when using the aforementioned biased cost metric, and when considering a standard implementation of Ekerå–Håstad's algorithm with windowing but no other significant optimizations. The same holds true for Ekerå–Gärtner's extensions of Regev's algorithm, for the general DLP in safe-prime groups — but not for the short DLP in such groups, or for the DLP in Schnorr groups, which are arguably the two cryptographically relevant problems to consider.

Hence, if we envisage non-computational quantum memory to be cheap, then Regev's factoring algorithm may have an advantage over the existing state-of-the-art algorithms for cryptographically relevant instances of the RSA IFP. Note however that even in our biased cost metric, where we ignore the space usage, there is only a per-run advantage — not an overall advantage.

A key reason for why the existing state-of-the-art variations of Shor's algorithm outperform Regev's algorithm in our comparisons is that many of the techniques used to derive and optimize these variations do not translate directly to Regev's algorithm. Regev's algorithm is still quite new, however, and more optimizations may become available as time progresses. A more detailed cost comparison may then be warranted to establish which algorithm has the advantage, but for now, the above biased comparison suffices, when not considering space to be cheap.

Acknowledgements

We are grateful to Johan Håstad for his comments on early versions of this manuscript, and to Oded Regev's for his generous support. We thank Craig Gidney, Seyoon Ragavan and Vinod Vaikuntanathan for useful discussions. We furthermore thank Seyoon Ragavan for pointing out that it may in some cases be beneficial to allow d to grow past the point where large modular multiplications are required to compute the c_j.

This work was supported by the KTH Centre for Cyber Defence and Information Security (CDIS), and funded and supported by the Swedish NCSA that is a part of the Swedish Armed Forces. Computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at PDC at KTH partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

[9]The per-run cost of Regev's original algorithm and Ekerå–Gärtner's extensions without the space-saving optimizations is $4 \log D$ in our metric, for D as in the tables in App. A. As previously noted, Ragavan [Rag24] discusses an option for reducing the cost from $4 \log D$ to $(2 + \epsilon) \log D$.

References

[ACD$^+$24] M.R. Albrecht, B. Curtis, L. Ducas, F. Göpfert, H. Hunt, H. Kippen, C. Lefebvre, J. Owen, R. Player, L. Pulles, M. Schmidt, S. Scott, F. Virdia, M. Walter, and C. Yun. Security estimates for lattice problems. GitHub repository malb/lattice-estimator, 2014–2024. URL: https://github.com/malb/lattice-estimator.

[APS15] M.R. Albrecht, R. Player, and S. Scott. On the concrete hardness of Learning with Errors. *J. Math. Cryptol.*, 9(3):169–203, 2015. doi:10.1515/jmc-2015-0016.

[BBP24] R. Barbulescu, M. Barcau, and V. Paşol. A comprehensive analysis of regev's quantum algorithm. Cryptology ePrint Archive, Paper 2024/1758, 2024. (Dated 2024-11-05.). URL: https://eprint.iacr.org/2024/1758.

[BCR$^+$18] E. Barker, L. Chen, A. Roginsky, A. Vassilev, and R. Davis. Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. National Institute of Standards and Technology (NIST) Special Publication (SP) 800-56A, 2018. (3rd revision). doi:10.6028/NIST.SP.800-56Ar3.

[BCR$^+$19] E. Barker, L. Chen, A. Roginsky, A. Vassilev, R. Davis, and S. Simon. Recommendation for pair-wise key-establishment using integer factorization cryptography. National Institute of Standards and Technology (NIST) Special Publication (SP) 800-56B, 2019. (2nd revision). doi:10.6028/NIST.SP.800-56Br2.

[Che13] Y. Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Université Paris Diderot (Paris 7), 2013. URL: http://www.theses.fr/2013PA077242.

[Cop02] D. Coppersmith. An approximate Fourier transform useful in quantum factoring, 2002. (Also IBM Research Report RC 19642.). arXiv:quant-ph/0201067v1.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976. doi:10.1109/TIT.1976.1055638.

[EG24a] M. Ekerå and J. Gärtner. Simulating Regev's quantum factoring algorithm and Ekerå–Gärtner's extensions to discrete logarithm finding, order finding and factoring via order finding. GitHub repository ekera/regevnum, 2023–2024. URL: https://github.com/ekera/regevnum.

[EG24b] M. Ekerå and J. Gärtner. Extending Regev's factoring algorithm to compute discrete logarithms. In M.-J. Saarinen and D. Smith-Tone, editors, *Post-Quantum Cryptography*, pages 211–242, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-62746-0_10.

[EH17] M. Ekerå and J. Håstad. Quantum algorithms for computing short discrete logarithms and factoring RSA integers. In T. Lange and T. Takagi, editors, *Post-Quantum Cryptography*, pages 347–363, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-59879-6_20.

[Eke20] M. Ekerå. On post-processing in the quantum algorithm for computing short discrete logarithms. *Des. Codes Cryptogr.*, 88(11):2313–2335, 2020. doi:10.1007/s10623-020-00783-2.

[Eke21] M. Ekerå. Quantum algorithms for computing general discrete logarithms and orders with tradeoffs. *J. Math. Cryptol.*, 15(1):359–407, 2021. doi:10.1515/jmc-2020-0006.

[Eke23] M. Ekerå. On the success probability of the quantum algorithm for the short DLP, 2023. arXiv:2309.01754v1.

[Eke24a] M. Ekerå. Qunundrum. GitHub repository ekera/qunundrum, 2020–2024. URL: https://github.com/ekera/qunundrum.

[Eke24b] M. Ekerå. Revisiting Shor's quantum algorithm for computing general discrete logarithms, 2024. arXiv:1905.09084v4.

[GE21] C. Gidney and M. Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021. doi:10.22331/q-2021-04-15-433.

[Gid19] C. Gidney. Windowed quantum arithmetic, 2019. arXiv:1905.07682v1.

[Gil16] D. Gillmor. Negotiated finite field Diffie–Hellman ephemeral parameters for Transport Layer Security (TLS). Request for Comments (RFC) 7919, 2016. doi:10.17487/RFC7919.

[KJ17] B. S. Kaliski Jr. Targeted Fibonacci exponentiation, 2017. arXiv:1711.02491.

[KK03] M. Kojo and T. Kivinen. More modular exponential (MODP) Diffie–Hellman groups for Internet Key Exchange (IKE). Request for Comments (RFC) 3526, 2003. doi:10.17487/RFC3526.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982. doi:10.1007/BF01457454.

[ME99] M. Mosca and A. Ekert. The hidden subgroup problem and eigenvalue estimation on a quantum computer. In C.P. Williams, editor, *Quantum Computing and Quantum Communications*, pages 174–188. Springer Berlin Heidelberg, 1999. doi:10.1007/3-540-49208-9_15.

[Nat94] National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). Federal Information Processing Standard (FIPS) 186, 1994. doi:10.6028/NIST.FIPS.186.

[Nat23] National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS). Federal Information Processing Standard (FIPS) 186-5, 2023. doi:10.6028/NIST.FIPS.186-5.

[NC23] National Institute of Standards and Technology (NIST) and Canadian Centre for Cyber Security (CCCS). Implementation guidance for FIPS 140-2 and the cryptographic module validation program, 2023. (Dated October 30, 2023.).

[Pil24] C. Pilatte. Unconditional correctness of recent quantum algorithms for factoring and computing discrete logarithms, 2024. arXiv:2404.16450v1.

[PP00] S. Parker and M.B. Plenio. Efficient factorization with a single pure qubit and $\log n$ mixed qubits. *Phys. Rev. Lett.*, 85:3049–3052, 2000. doi:10.1103/PhysRevLett.85.3049.

[Rag24] S. Ragavan. Regev factoring beyond Fibonacci: Optimizing prefactors. Cryptology ePrint Archive, Paper 2024/636, 2024. (Dated 2024-07-01.). URL: https://eprint.iacr.org/2024/636.

[Reg25] O. Regev. An efficient quantum factoring algorithm. *J. ACM*, 72(1):10:1–13, 2025. doi:10.1145/3708471.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. doi:10.1145/359340.359342.

[RV23] S. Ragavan and V. Vaikuntanathan. Optimizing space in Regev's factoring algorithm, 2023. arXiv:2310.00899v1.

[RV24] S. Ragavan and V. Vaikuntanathan. Space-efficient and noise-robust quantum factoring. In L. Reyzin and D. Stebila, editors, *Advances in Cryptology — CRYPTO 2024*, pages 107–140, Cham, Switzerland, 2024. Springer Nature. doi:10.1007/978-3-031-68391-6_4.

[SE94] C.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66(1):181–199, 1994. doi:10.1007/BF01581144.

[Sei01] J.-P. Seifert. Using fewer qubits in Shor's factorization algorithm via simultaneous Diophantine approximation. In D. Naccache, editor, *Topics in Cryptology — CT-RSA 2001*, pages 319–327, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. doi:10.1007/3-540-45353-9_24.

[Sho94] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi:10.1109/SFCS.1994.365700.

[Sho97] P.W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. doi:10.1137/S0097539795293172.

[VM08] R. Van Meter. *Architecture of a Quantum Multicomputer Optimized for Shor's Factoring Algorithm*. PhD thesis, Keio University, 2008.

[VMI05] R. Van Meter and K.M. Itoh. Fast quantum modular exponentiation. *Phys. Rev. A*, 71:052320, 2005. doi:10.1103/PhysRevA.71.052320.

A Tables

In this appendix, we tabulate cost comparisons between Regev's variation [Reg25] of Shor's algorithms [Sho94, Sho97], and Ekerå–Gärtner's extensions [EG24b] thereof, on the one hand, and Ekerå–Håstad's [EH17] and Ekerå's [Eke24b] variations of Shor's algorithms [Sho94, Sho97] on the other. In the tables, we inherit notation from the aforementioned works, sometimes leading to the same symbol being used to denote different quantities depending on where in the table it is used. To prevent misunderstandings, we describe how the tables were constructed below:

- **For Regev's algorithm and Ekerå–Gärtner's extensions (EGR):**

  As in [Reg25, EG24b], we let $n = \lceil \log N \rceil$. We let d denote the dimension, and m the number of runs. We let $R = 2^{C\sqrt{n}}$, for $C > 0$, and $D = 2^{\lceil \log(2\sqrt{d}R) \rceil}$.

  We use Ragavan's generalization [Rag24] of the space-saving optimizations of Ragavan and Vaikuntanathan [RV24]. We let r and s be as defined by Ragavan [Rag24].

  We let $K^{(r)}$ be maximal such that $G^{(r)}_{K^{(r)}} \leq D$ as in [Rag24, Sect. 3.2]. If $r > 1$, we tabulate $K^{(r)}$ and r. Otherwise, if $r = 1$, we only tabulate $K = K^{(1)}$.

  We do not tabulate s. Instead, we pick s as function of r: If $r = 1$, we must pick $s = 1$. Otherwise, if $r > 1$ and a power of two, we pick $s = r/2$. Otherwise, we pick s as the greatest power of two that divides r. This is the optimal way to pick s in our cost metric as it minimizes $f(r, s)$, see Sect. 3.2.2.

  The number of large multiplications modulo N of the form

  $$| \, u, v, t, 0^S \, \rangle \to | \, u, v, (t + uv) \bmod N, 0^S \, \rangle \qquad (9)$$

  that need to be performed per run is $(2^3 \cdot M(d, r) + f(r, s)) \cdot K^{(r)}$, see Sect. 3.5. Overall, the number of large multiplications modulo N that need to be performed is hence $(2^3 \cdot M(d, r) + f(r, s)) \cdot K^{(r)} \cdot m$. We tabulate these counts in the columns denoted "#ops". (In Tab. 6, where $m \to \infty$, we only tabulate the per-run count.)

  For reference, we also include $M(d, r)$ in the tables, in the column denoted M, except in Tabs. 1–4 as $M(d, r)$ would always be zero in these tables.

- **For Ekerå–Håstad's variation of Shor's algorithm (EHS):**

  As in [EH17], we let m be an upper bound on the bit length of the short discrete logarithm d. For the RSA IFP, we have that $m = \lceil \log N \rceil / 2 - 1$, whereas $m = 2z$ for the DLP in safe-prime groups, for z the strength level in bits.

  We consider both the case of solving for d in a single run via [Eke23] with $\Delta = 30$ and $\ell = m - \Delta$, and of making tradeoffs with tradeoff factor $s > 1$ and solving via [Eke20] in $n \geq s$ runs with $\ell = \lceil m/s \rceil$.

  In the former case, the success probability $\ggg 99\%$, see [Eke23, Tab. 1]. In the latter case, we pick s and n from [Eke20, Tab. 3], leading to a success probability $\geq 99\%$ without enumerating. Other parameterizations are of course possible.

  We use standard arithmetic, with windowing with a $w = 10$ bit window, and without windowing corresponding to having a $w = 1$ bit window.

  The total exponent length per run is $m + 2\ell$. It follows that number of large multiplications modulo N of the form (9) that need to be performed per run is $2 \cdot \lceil (m + 2\ell)/w \rceil$. Overall, the number of such operations that need to be performed is thus $2n \cdot \lceil (m + 2\ell)/w \rceil$. We tabulate these counts in the columns denoted "#ops".

  We define the advantage as the quotient between the number of such operations required by Regev's algorithm, or EGR, and EHS, respectively. We tabulate the advantage, per run and overall, in the columns denoted "adv". (In Tab. 6, we tabulate only the advantage per run.)

- **For Ekerå's variation of Shor's algorithm (ES):**

  As in [Eke24b], we let m be an upper bound on the bit length of the logarithm d. For the general DLP in safe-prime groups we have that $m = \lceil \log N \rceil - 1$. For the DLP in Schnorr groups we instead have that $m = 2z$, for z the strength level in bits.

  We consider both the case of solving for d in a single run via [Eke24b, Sect. 6.1] with $\varsigma = 0$ and $\ell = m$, and of making tradeoffs with tradeoff factor $s > 1$ and solving via [Eke24b, Sect. 6.2] in $n \geq s$ runs with $\ell = \lceil m/s \rceil$.

  In the former case, the success probability $\ggg 99\%$, provided that one performs a limited search in the classical post-processing, see [Eke23, Tab. 1 in App. A.1]. In the latter case, we pick s, ς and n so as to achieve a reasonable tradeoff and a success probability $\geq 99\%$ without enumerating, and with $\eta_1 = \ldots = \eta_n = 0$. This based on simulations performed with the Qunundrum [Eke24a] suite of MPI programs on the Dardel HPE Cray EX supercomputer at PDC at KTH. Other parameterizations are of course possible.

  We use standard arithmetic, with windowing with a $w = 10$ bit window.

  The total exponent length per run is $m + \varsigma + \ell$. It follows that number of large multiplications modulo N of the form (9) that need to be performed per run is $2 \cdot \lceil (m + \varsigma + \ell)/w \rceil$. Overall, the number of such operations that need to be performed is thus $2n \cdot \lceil (m + \varsigma + \ell)/w \rceil$. We tabulate these counts in the columns denoted "#ops".

  We define the advantage as the quotient between the number of such operations required by Regev's algorithm, or EGR, and ES, respectively. We tabulate the advantage, per run and overall, in the columns denoted "adv".

  It should be noted that the analysis in [Eke24b] is heuristic. We could instead have based our comparison on the algorithm and analysis in [Eke21] that does not require the group order to be known. This would have led to a cost profile essentially identical to that for EHS [EH17, Eke20, Eke23], but for the fact that somewhat smaller tradeoff factors s would have had to be selected for the Schnorr groups to ensure that the upper bound on the approximation error in the analysis in [Eke21] is sufficiently low.

A.1 RSA IFP

A.1.1 A basic baseline comparison

Table 1: Comparison between Regev's algorithm [Reg25] (with [RV24, Rag24], LLL, $r = 1$, $d = \lceil \sqrt{n} \rceil$, $m = d + 4$) and Ekerå–Håstad's variation [EH17, Eke20, Eke23] of Shor's algorithm [Sho94, Sho97] for the RSA IFP (with $w = 1$). For further information on this table and how to interpret it, see Sect. 4.1.1 and App. A.

	IFP via Regev [Reg25] with [RV24, Rag24]							**RSA IFP via Ekerå–Håstad [EH17, Eke20, Eke23]**							
						per run	**overall**					**per run**		**overall**	
$\lceil \log N \rceil$	d	m	C	$\log D$	K	#ops	#ops	m	s	ℓ	n	#ops	adv	#ops	adv
2048	46	50	2.03	96	138	2760	138000	1023	–	993	1	6018	0.46	6018	22.9
									17	61	20	2290	1.20	45800	3.01
3072	56	60	2.05	118	170	3400	204000	1535	–	1505	1	9090	0.37	9090	22.4
									21	74	24	3366	1.01	80784	2.52
4096	64	68	2.08	138	199	3980	270640	2047	–	2017	1	12162	0.33	12162	22.2
									24	86	27	4438	0.90	119826	2.25
6144	79	83	2.07	167	241	4820	400060	3071	–	3041	1	18306	0.26	18306	21.8
									31	100	34	6542	0.74	222428	1.79
8192	91	95	2.08	193	278	5560	528200	4095	–	4065	1	24450	0.23	24450	21.6
									34	121	37	8674	0.64	320938	1.64

A.1.2 Using LLL and $r = 1$

Table 2: Comparison between Ekerå–Gärtner's extension [EG24b] of Regev's algorithm [Reg25] (with [RV24, Rag24], LLL, $r = 1$, optimal d and m) and Ekerå–Håstad's variation [EH17, Eke20, Eke23] of Shor's algorithm [Sho94, Sho97] (with $w = 10$) for the RSA IFP. For further information on this table and how to interpret it, see Sect. 4.1.2 and App. A.

	IFP via Regev [Reg25] with [RV24, Rag24, EG24b]							**RSA IFP via Ekerå–Håstad [EH17, Eke20, Eke23]**							
						per run	**overall**					**per run**		**overall**	
$\lceil \log N \rceil$	d	m	C	$\log D$	K	#ops	#ops	m	s	ℓ	n	#ops	adv	#ops	adv
2048	137	181	1.02	51	74	1480	267880	1023	–	993	1	602	2.45	602	444
									17	61	20	230	6.43	4600	58.2
3072	182	222	1.01	61	88	1760	390720	1535	–	1505	1	910	1.93	910	429
									21	74	24	338	5.20	8112	48.1
4096	210	256	1.01	70	101	2020	517120	2047	–	2017	1	1218	1.65	1218	424
									24	86	27	444	4.54	11988	43.1
6144	237	314	1.02	85	123	2460	772440	3071	–	3041	1	1832	1.34	1832	421
									31	100	34	656	3.75	22304	34.6
8192	297	362	1.01	97	140	2800	1013600	4095	–	4065	1	2446	1.14	2446	414
									34	121	37	868	3.22	32116	31.5

A.1.3 Using BKZ-200 and $r = 1$

Table 3: Comparison between Ekerå–Gärtner's extension [EG24b] of Regev's algorithm [Reg25] (with [RV24, Rag24], BKZ-200, $r = 1$, optimal d and m) and Ekerå–Håstad's variation [EH17, Eke20, Eke23] of Shor's algorithm [Sho94, Sho97] (with $w = 10$) for the RSA IFP. For further information on this table and how to interpret it, see Sect. 4.1.3 and App. A.

	IFP via Regev [Reg25] with [RV24, Rag24, EG24b]							**RSA IFP via Ekerå–Håstad [EH17, Eke20, Eke23]**							
						per run	**overall**					**per run**		**overall**	
$\lceil \log N \rceil$	d	m	C	$\log D$	K	#ops	#ops	m	s	ℓ	n	#ops	adv	#ops	adv
2048	232	342	0.55	30	43	860	294120	1023	–	993	1	602	1.42	602	488
									17	61	20	230	3.73	4600	63.9
3072	284	419	0.55	36	52	1040	435760	1535	–	1505	1	910	1.14	910	478
									21	74	24	338	3.07	8112	53.7
4096	366	483	0.54	40	58	1160	560280	2047	–	2017	1	1218	0.95	1218	460
									24	86	27	444	2.61	11988	46.7
6144	449	592	0.54	48	69	1380	816960	3071	–	3041	1	1832	0.75	1832	445
									31	100	34	656	2.10	22304	36.6
8192	652	683	0.53	54	78	1560	1065480	4095	–	4065	1	2446	0.64	2446	435
									34	121	37	868	1.79	32116	33.1

A.1.4 Using LLL and optimal r

Table 4: Comparison between Ekerå–Gärtner's extension [EG24b] of Regev's algorithm [Reg25] (with [RV24, Rag24], LLL, optimal r, optimal d and m) and Ekerå–Håstad's variation [EH17, Eke20, Eke23] of Shor's algorithm [Sho94, Sho97] (with $w = 10$) for the RSA IFP. For further information on this table and how to interpret it, see Sect. 4.1.4 and App. A.

	IFP via Regev [Reg25] with [RV24, Rag24, EG24b]								**RSA IFP via Ekerå–Håstad [EH17, Eke20, Eke23]**							
							per run	**overall**					**per run**		**overall**	
$\lceil \log N \rceil$	d	m	C	$\log D$	$K^{(r)}$	r	#ops	#ops	m	s	ℓ	n	#ops	adv	#ops	adv
2048	78	181	1.19	58	28	4	952	172312	1023	–	993	1	602	1.58	602	286
										17	61	20	230	4.13	4600	37.4
3072	107	222	1.14	68	33	4	1122	249084	1535	–	1505	1	910	1.23	910	273
										21	74	24	338	3.31	8112	30.7
4096	138	256	1.10	75	36	4	1224	313344	2047	–	2017	1	1218	1.00	1218	257
										24	86	27	444	2.75	11988	26.1
6144	211	314	1.04	87	42	4	1428	448392	3071	–	3041	1	1832	0.78	1832	244
										31	100	34	656	2.17	22304	20.1
8192	195	362	1.10	105	40	6	1600	579200	4095	–	4065	1	2446	0.65	2446	236
										34	121	37	868	1.84	32116	18.0

A.1.5 Using BKZ-200 and optimal r

Table 5: Comparison between Ekerå–Gärtner's extension [EG24b] of Regev's algorithm [Reg25] (with [RV24, Rag24], BKZ-200, optimal r, optimal d and m) and Ekerå–Håstad's variation [EH17, Eke20, Eke23] of Shor's algorithm [Sho94, Sho97] (with $w = 10$) for the RSA IFP. For further information on this table and how to interpret it, see Sect. 4.1.5 and App. A.

	IFP via Regev [Reg25] with [RV24, Rag24, EG24b]									**RSA IFP via Ekerå–Håstad [EH17, Eke20, Eke23]**							
								per run	**overall**					**per run**		**overall**	
$\lceil \log N \rceil$	d	m	C	$\log D$	$K^{(r)}$	r	M	#ops	#ops	m	s	ℓ	n	#ops	adv	#ops	adv
2048	137	342	0.65	34	27	2	0	702	240084	1023	–	993	1	602	1.16	602	398
											17	61	20	230	3.05	4600	52.1
3072	196	419	0.61	39	19	4	1	798	334362	1535	–	1505	1	910	0.88	910	367
											21	74	24	338	2.36	8112	41.2
4096	148	483	0.74	52	25	4	0	850	410550	2047	–	2017	1	1218	0.70	1218	337
											24	86	27	444	1.91	11988	34.2
6144	201	592	0.70	60	29	4	0	986	583712	3071	–	3041	1	1832	0.54	1832	318
											31	100	34	656	1.50	22304	26.1
8192	255	683	0.67	66	32	4	0	1088	743104	4095	–	4065	1	2446	0.44	2446	303
											34	121	37	868	1.25	32116	23.1

A.1.6 Using perfect reduction and optimal r

Table 6: Comparison between Regev's algorithm [Reg25] (with [RV24, Rag24], perfect reduction, optimal r, optimal d and $m \to \infty$) and Ekerå–Håstad's variation [EH17, Eke20, Eke23] of Shor's algorithm [Sho94, Sho97] (with $w = 10$) for the RSA IFP. Note that this table is special in that it considers Regev's algorithm both with and without the extensions of Ekerå and Gärtner [EG24b], and in that the number of runs $m \to \infty$ for Regev giving Ekerå–Håstad an infinite overall advantage. Note furthermore that since C is quite small in this table, we exceptionally tabulate C with precision 0.001. For further information on this table and how to interpret it, see Sect. 4.1.6 and App. A.

	IFP via Regev [Reg25] with [RV24, Rag24, EG24b]							**RSA IFP via Ekerå–Håstad [EH17, Eke20]**					
							per run					**per run**	
$\lceil \log N \rceil$	d	C	$\log D$	K	r	M	#ops	m	s	ℓ	n	#ops	adv
2048	143	0.317	19	15	2	1	510	1023	17	61	20	230	2.21
	227	0.200	14	11	2	1	374						1.62
3072	204	0.272	20	16	2	1	544	1535	21	74	24	338	1.60
	351	0.158	14	11	2	1	374						1.10
4096	256	0.250	21	17	2	1	578	2047	24	86	27	444	1.30
	427	0.150	15	12	2	1	408						0.92
6144	347	0.226	23	18	2	1	612	3071	31	100	34	656	0.93
	454	0.173	19	11	3	1	440						0.67
8192	809	0.112	16	23	1	1	644	4095	34	121	37	868	0.74
	809	0.112	16	13	2	1	442						0.51

A.2 DLP in finite fields

A.2.1 General DLP in safe-prime groups

Table 7: Comparison between Ekerå–Gärtner's extension [EG24b] of Regev's algorithm [Reg25] (with [RV24, Rag24], BKZ-200, optimal r, optimal d and m) and Ekerå's variation [Eke24b] of Shor's algorithm [Sho94, Sho97] (with $w = 10$) for the general DLP in safe-prime groups. For further information on this table and how to interpret it, see Sect. 4.2.1 and App. A.

		DLP via Ekerå–Gärtner [EG24b, Reg25] with [RV24, Rag24]									**DLP via Ekerå's variation [Eke24b] of Shor [Sho94, Sho97]**								
									per run	**overall**						**per run**		**overall**	
$\lceil \log N \rceil$	z	d	m	C	$\log D$	$K^{(r)}$	r	M	#ops	#ops	m	s	ς	ℓ	n	#ops	adv	#ops	adv
2048	112	137	342	0.65	34	27	2	0	710	242820	2047	1	0	2047	1	820	0.86	820	296
												24	11	86	27	430	1.65	11610	20.9
3072	128	196	419	0.61	39	19	4	1	806	337714	3071	1	0	3071	1	1230	0.66	1230	274
												31	12	100	34	638	1.26	21692	15.5
4096	152	148	483	0.74	52	25	4	0	862	416346	4095	1	0	4095	1	1638	0.53	1638	254
												34	12	121	37	846	1.01	31302	13.3
6144	176	201	592	0.70	60	29	4	0	998	590816	6143	1	0	6143	1	2458	0.41	2458	240
												37	12	167	40	1266	0.79	50640	11.6
8192	200	255	683	0.67	66	32	4	0	1102	752666	8191	1	0	8191	1	3278	0.34	3278	229
												40	12	205	43	1682	0.66	72326	10.4

A.2.2 Short DLP in safe-prime groups

Table 8: Comparison between Ekerå–Gärtner's extension [EG24b] of Regev's algorithm [Reg25] (with [RV24, Rag24], BKZ-200, optimal r, optimal d and m) and Ekerå–Håstad's variation [EH17, Eke20, Eke23] of Shor's algorithm [Sho94, Sho97] (with $w = 10$) for the short DLP in safe-prime groups. For further information on this table and how to interpret it, see Sect. 4.2.2 and App. A.

		DLP via Ekerå–Gärtner [EG24b, Reg25] with [RV24, Rag24]									**short DLP via Ekerå–Håstad [EH17, Eke20, Eke23]**							
									per run	**overall**					**per run**		**overall**	
$\lceil \log N \rceil$	z	d	m	C	$\log D$	$K^{(r)}$	r	M	#ops	#ops	m	s	ℓ	n	#ops	adv	#ops	adv
2048	112	137	342	0.65	34	27	2	0	710	242820	224	–	194	1	124	5.72	124	1950
												7	32	10	58	12.2	580	418
3072	128	196	419	0.61	39	19	4	1	806	337714	256	–	226	1	142	5.67	142	2370
												8	32	11	64	12.5	704	479
4096	152	148	483	0.74	52	25	4	0	862	416346	304	–	274	1	172	5.01	172	2420
												9	34	12	76	11.3	912	456
6144	176	201	592	0.70	60	29	4	0	998	590816	352	–	322	1	200	4.99	200	2950
												10	36	13	86	11.6	1118	528
8192	200	255	683	0.67	66	32	4	0	1102	752666	400	–	370	1	228	4.83	228	3300
												11	37	14	96	11.4	1344	560

A.2.3 DLP in Schnorr groups

Table 9: Comparison between Ekerå–Gärtner's extension [EG24b] of Regev's algorithm [Reg25] (with [RV24, Rag24], BKZ-200, optimal r, optimal d and m) and Ekerå's variation [Eke24b] of Shor's algorithm [Sho94, Sho97] (with $w = 10$) for the DLP in Schnorr groups. For further information on this table and how to interpret it, see Sect. 4.2.3 and App. A.

		DLP via Ekerå–Gärtner [EG24b, Reg25] with [RV24, Rag24]									**DLP via Ekerå's variation [Eke24b] of Shor [Sho94, Sho97]**								
									per run	**overall**						**per run**		**overall**	
$\lceil \log N \rceil$	z	d	m	C	$\log D$	$K^{(r)}$	r	M	#ops	#ops	m	s	ς	ℓ	n	#ops	adv	#ops	adv
2048	112	137	342	0.65	34	27	2	0	710	242820	224	1	0	224	1	90	7.88	90	2690
												7	9	32	10	54	13.1	540	449
3072	128	196	419	0.61	39	19	4	1	806	337714	256	1	0	256	1	104	7.75	104	3240
												8	9	32	11	60	13.4	660	511
4096	152	148	483	0.74	52	25	4	0	862	416346	304	1	0	304	1	122	7.06	122	3410
												9	10	34	12	70	12.3	840	495
6144	176	201	592	0.70	60	29	4	0	998	590816	352	1	0	352	1	142	7.02	142	4160
												10	10	36	13	80	12.4	1040	568
8192	200	255	683	0.67	66	32	4	0	1102	752666	400	1	0	400	1	160	6.88	160	4700
												11	10	37	14	90	12.2	1260	597