

Implementation And Analysis of Regev's Quantum Factorization Algorithm

Przemysław Pawlitko, Natalia Moćko, Marcin Niemiec * and Piotr Cholda

AGH University of Krakow, Institute of Telecommunications and Cybersecurity, Mickiewicza 30,
30-059 Krakow, Poland

* Corresponding author: niemiec@agh.edu.pl

Received date: 20 November 2025; Accepted date: 10 March 2026; Published online: 17 March 2026

Abstract: Quantum computing represents a significant advancement in computational capabilities. Of particular concern is its impact on asymmetric cryptography through, notably, Shor's algorithm and the more recently developed Regev's algorithm for factoring composite numbers. We present our implementation of the latter with the aim of exposing experimental phenomena. Our analysis encompasses both quantum simulation results and classical component examples, with particular emphasis on comparative cases between Regev's and Shor's algorithms. Our experimental results reveal that Regev's algorithm outperforms Shor's algorithm for certain composite numbers in practice. However, we observed significant performance variations across different input values. Despite Regev's algorithm's theoretical asymptotic advantage in effectiveness, our implementation exhibited longer execution times than Shor's algorithm for small-integer factorization in both quantum and classical components. These findings offer insights into the practical challenges and performance characteristics of implementing Regev's algorithm in realistic quantum computing scenarios.

Keywords: factorization; quantum algorithms; quantum computers; Regev's algorithm; Shor's algorithm

1. Introduction

Public-key (asymmetric) cryptography is a fundamental pillar of modern cybersecurity. Its principles underpin the security of numerous systems, including secure Internet communication, authentication protocols, banking transactions, cloud data protection, software integrity verification, medical data encryption, and many other applications. Without secure public-key cryptography, sensitive information would be exposed, privacy compromised, and the risk of online fraud and cybercrime would increase significantly. One of the most widely used public-key algorithms, RSA [1], is based on the mathematical difficulty of factorizing a large semiprime integer N , which is the product of two prime numbers p and q . For classical computers, factorizing such a number is believed to be computationally infeasible within a practical (i.e., polynomial) timeframe. For example, breaking RSA encryption with a 2048-bit key by computing 1.6×10^{16} operations per second would require approximately 19.8 quadrillion years using the brute-force method on classical computers. Moreover, the compromise of RSA would have far-reaching implications for other cryptographic algorithms that are based on similar computational-problem principles (i.e., those based on discrete logarithms over finite fields).

Although breaking RSA encryption is currently impossible for classical computers, in 1994 the American computer scientist and mathematician Peter Shor published a work "*Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*" [2,3] that proved that factorizing large numbers efficiently is possible using quantum computers. His algorithm had profound implications for cryptography, raising awareness that current public-key encryption methods would not remain secure in the era of advanced quantum computing [4]. It is worth noting that other approaches to factorization, e.g., based on optimization approaches, are also proposed [5].

However, the development of scalable and powerful quantum computers capable of running Shor's algorithm remains a significant technical challenge. As the number of qubits and quantum gates increases, quantum computers become more susceptible to quantum decoherence, as described by Mutter and Burkard in *Theory of qubit noise characterization using the long-time cavity transmission* [6]. Decoherence disrupts quantum effects, which are essential for the proper execution of quantum algorithms. Shor's algorithm, in particular, requires a large number of quantum gates, making it desirable to minimize this requirement to improve the feasibility of practical implementations.

To address these challenges, Oded Regev proposed an alternative quantum algorithm in his recent work, “*An Efficient Quantum Factoring Algorithm*” [7] (we cite only the last, i.e., journal, version of the work, but the first version was published in 2023 in arxiv), which can be thought of as a multidimensional extension of Shor’s algorithm. Regev’s algorithm reduces the number of quantum gates required for factorization compared to Shor’s algorithm. However, Regev acknowledges that it remains unclear whether this reduction will translate into practical improvements in the physical implementation of quantum computers. Despite this uncertainty, his work represents a critical step toward overcoming the obstacles associated with quantum computing and advancing the field of quantum cryptography.

In 2024, Kiebert provided a detailed analysis of this algorithm in *Oded Regev’s Quantum Factoring Algorithm* [8]. His work presents the mathematical foundations and key theorems required for a deep understanding of Regev’s algorithm. This work provided the necessary details for the classical part of our implementation, guiding both the interpretation of the underlying mathematics and its subsequent realization in code.

Given that Shor’s algorithm was introduced over two decades ago, it has benefited from extensive research devoted to its optimization. In contrast, Regev’s algorithm is a relatively recent development, having been proposed only about two years ago. This disparity in maturity between the two algorithms significantly influences the extent to which they have been optimized. Ragavan and Vaikuntanathan present a thorough comparison of optimized variants of Shor’s and Regev’s algorithms [9]. Furthermore, in 2025, Ekerå and Gärtner published a high-level comparison of factoring algorithms that included both Shor’s and Regev’s approaches [10].

The authors of this paper built an implementation of Regev’s algorithm based on Stepien’s implementation of Shor’s algorithm [11]. In his work, the author implemented from scratch and compared the effectiveness of Shor’s algorithm using four different modular exponentiation gates. These gates were created by Beauregard [12], Takahashi [13], and Häner [14]. The fourth gate is Stepien’s own combination of the gates mentioned above. Stepien implemented each of the more complicated quantum gates using elementary quantum gates. We treated his implementations as building blocks and used the Häner variant of the modular exponentiation gate in our own quantum circuit.

To execute Regev’s algorithm on the constructed quantum circuit, we employed a simulated environment without decoherence, which is described in detail in Section 3. We also attempted to run the algorithm on a cloud-accessible quantum computer; however, at the time of the experiments, the executions consistently resulted in timeouts without any error notifications. We therefore suspect that the gate complexity of the algorithm exceeds the practical capabilities of currently available cloud-based quantum hardware.

1.1. Contribution of the Paper

At the time of working on this paper, no publicly available implementation of Regev’s quantum algorithm for quantum computers existed. Therefore, the main challenge was to develop the first implementation of Regev’s algorithm for quantum computers. The implementation served as the foundation for analyzing experimentally observable phenomena related to its speed and effectiveness, achieved by introducing and varying relevant parameters. Throughout our study, comparisons were made between Regev’s algorithm and Shor’s algorithm to highlight their respective strengths and limitations. Additionally, this paper aimed to identify and present potential areas for future research on Regev’s algorithm, providing a starting point for subsequent studies in the field.

1.2. Structure of the Paper

The remainder of this paper is organized as follows. First, in Section 2, both Shor’s and Regev’s algorithms are reviewed and briefly compared, especially from the viewpoint of feasible implementations of the quantum part. The next part, Section 3, focuses on the non-trivial details of Regev’s algorithm implementation. Section 4 elaborates on the performance comparison of Regev’s and Shor’s algorithms, although the results related to the former are emphasized as more original. Finally, Section 5 concludes the paper.

2. Basics of Shor’s and Regev’s Algorithms

Both algorithms aim to factorize a semiprime N , an n -bit integer that is the product of prime numbers $p, q \in \mathbb{P}$ (i.e., $N = p \cdot q$). However, the factorization process is carried out in different ways.

2.1. Idea of the Algorithms

Basically, Shor's algorithm finds the smallest even period r of a modular exponentiation function, which is defined as follows:

$$z \mapsto a^z \bmod N.$$

Here, a is a random integer that belongs to the group of units of $(\mathbb{Z}/N\mathbb{Z})^\times$ and $z \in \mathbb{Z}$. After finding the period of this function, factorizing N is trivial and begins with this equation:

$$a^r \equiv 1 \bmod N.$$

During the next steps, we use the condition that r is an even number:

$$\begin{aligned} a^r - 1 &\equiv 0 \bmod N; \\ \left(a^{\frac{r}{2}} - 1\right)\left(a^{\frac{r}{2}} + 1\right) &\equiv 0 \bmod N; \\ \left(a^{\frac{r}{2}} - 1\right)\left(a^{\frac{r}{2}} + 1\right) &\equiv 0 \bmod (p \cdot q). \end{aligned}$$

Therefore, there must exist such $k \in \mathbb{Z}$, that:

$$\left(a^{\frac{r}{2}} - 1\right)\left(a^{\frac{r}{2}} + 1\right) = p \cdot q \cdot k.$$

It is very likely that p is a divisor of $\left(a^{\frac{r}{2}} - 1\right)$ and q is a divisor of $\left(a^{\frac{r}{2}} + 1\right)$. As discussed by Shor [3], this probability “is at least $1 - 1/2^{k-1}$, where k is the number of distinct odd prime factors of n ”, so in this case there is approximately a 50% chance. One possibility is that both p and q are factors of $\left(a^{\frac{r}{2}} + 1\right)$ or $\left(a^{\frac{r}{2}} - 1\right)$. Theoretically, there should not be any case where they are both factors of $\left(a^{\frac{r}{2}} - 1\right)$, but sometimes Shor's algorithm finds a multiple of the smallest period. Once such an even period r is obtained, the prime factors p and q are computed using the greatest common divisor as follows:

$$\begin{aligned} p &= \gcd\left(\left(a^{\frac{r}{2}} - 1\right), N\right); \\ q &= \gcd\left(\left(a^{\frac{r}{2}} + 1\right), N\right) = \frac{N}{p}. \end{aligned}$$

If it comes to the implementation, the most time-consuming part of the algorithm is a previously described modular exponentiation function:

$$z \mapsto a^z \bmod N.$$

The reason is that seeing the period of this function requires z to be large enough. For the n -bit integer N , z increases to 2^n . Moreover, this function needs to be applied in superposition inside the quantum part of the algorithm. In order to reduce a computational load, the *repeated squaring trick* is used. Thus, quantum part needs n multiplications of n -bit numbers, so roughly $\tilde{O}(n^2)$ gates. The Big- $\tilde{O}$ -tilde notation ($\tilde{O}$) is commonly used in describing quantum metrics. Unlike the classical Big- O notation, it disregards logarithmic factors. For example, $O(n \log n) = \tilde{O}(n)$. Despite this reduction in computational complexity, the numbers that need to be computed are very large.

There is also an even more serious problem. Quantum effects, such as superposition and entanglement, can be destroyed during the run of an algorithm. This is because the larger the number of entangled particles and / or the larger the number of gates in a quantum circuit, the more quantum effects are susceptible to noise and decoherence, leading to destruction [6]. For this reason, it is desirable to reduce the number of quantum gates as much as possible.

To address these challenges, Oded Regev proposed the approach of extending to a larger number of dimensions [7]. The concept of his algorithm bears similarities to Shor's algorithm, as it also focuses on finding the period of a specific function. However, the function is defined in a slightly different manner:

$$(z_1, z_2, \dots, z_d) \mapsto a_1^{z_1} \cdot a_2^{z_2} \cdot \dots \cdot a_d^{z_d} \bmod N. \quad (1)$$

Here, the numbers $a_1, a_2, \dots, a_d$ are the first squared primes (that is, 4, 9, 25, ...), N is still an integer of n bits that is being factorized, and $z_i \in \mathbb{Z}$. After finding the period vector $[r_1, r_2, \dots, r_d]$ of this function, factorization of N is trivial, and – similarly to the transformations described above – it starts with the following step:

$$a_1^{r_1} \cdot a_2^{r_2} \cdot \dots \cdot a_d^{r_d} \equiv 1 \pmod{N}.$$

Let us define $a_i = b_i^2$. Due to the fact that each a_i is a squared prime b_i :

$$\begin{aligned} b_1^{2r_1} \cdot b_2^{2r_2} \cdot \dots \cdot b_d^{2r_d} &\equiv 1 \pmod{N}; \\ b_1^{2r_1} \cdot b_2^{2r_2} \cdot \dots \cdot b_d^{2r_d} - 1 &\equiv 0 \pmod{N}; \\ (b_1^{r_1} \cdot b_2^{r_2} \cdot \dots \cdot b_d^{r_d} - 1)(b_1^{r_1} \cdot b_2^{r_2} \cdot \dots \cdot b_d^{r_d} + 1) &\equiv 0 \pmod{N}. \end{aligned}$$

Regev's algorithm relies on a number-theoretic heuristic assumption, as described in [7]. This assumption states that at least half of the period vectors, when applied to the analyzed cyclic function and followed by bilateral square rooting, produce non-trivial factors of 1 mod N , that is, factors that are neither 1 nor -1 . Consequently, this assumption ensures that, in at least half of the cases, $b_1^{r_1} \cdot b_2^{r_2} \cdot \dots \cdot b_d^{r_d} \notin \{1, -1\} \pmod{N}$. As a result, it avoids the trivial solutions $(b_1^{r_1} \cdot b_2^{r_2} \cdot \dots \cdot b_d^{r_d} - 1) = 0$ or $(b_1^{r_1} \cdot b_2^{r_2} \cdot \dots \cdot b_d^{r_d} + 1) = 0$, thereby enabling the computation of p and q , as detailed in the following discussion.

From this point, the reasoning aligns with that of Shor's algorithm:

$$\begin{aligned} p &= \gcd((b_1^{r_1} \cdot b_2^{r_2} \cdot \dots \cdot b_d^{r_d} - 1), N); \\ q &= \gcd((b_1^{r_1} \cdot b_2^{r_2} \cdot \dots \cdot b_d^{r_d} + 1), N) = \frac{N}{p}. \end{aligned}$$

In the next lines, we present a sketch of the proof that this approach will also result in a period vector finding. It is based on the brief proof description that Regev outlined in his work.

To find a period, there need to be at least two combinations of the vectors $(z_1, z_2, \dots, z_d)$ that give the same function results. Following this condition, a period vector can be constructed by analyzing these two vector values.

The analyzed transform given in equation (1) is a product of d squared primes raised to a power of up to $2^{n/d}$. Thus, the total number of results of the analyzed function is $(2^{n/d})^d = 2^n$.

Furthermore, period r of the analyzed function abides by this inequality:

$$r \leq \phi(N) = (p-1)(q-1) = pq - p - q + 1 < N = pq \leq 2^n - 1 < 2^n.$$

Combining these two facts and using the pigeon hole principle, there have to be two vectors $(z_1, z_2, \dots, z_d)$ giving the same function result and that is why the period can be found.

With this approach, each z_i goes up to $2^{n/d}$ and not up to 2^n as in Shor's approach, which reduces the scale of the computed numbers, and thus reduces the computational load. Moreover, it appears that the number of gates needed in the quantum part of the original Regev's algorithm is reduced to $\tilde{O}(n^{3/2})$, which reduces the noise and decoherence of quantum effects. In our case, the gate complexity is on the order of $\tilde{O}(n^{5/2} \log n)$, whereas the width of the quantum circuit is smaller, on the order of $\tilde{O}(n)$.

2.2. Complexity Comparison

Quantum algorithms are characterized using three metrics: *width*, *depth* and *gate complexity*. Their values help us assess runtime, simplicity of implementation, and resilience to decoherence, which directly affect the effectiveness of the quantum algorithm. Moreover, the complexities of Regev's and Shor's algorithms are heavily influenced by the choice of multiplication algorithm. Depending on it, quantum metrics change significantly.

- *Width* metric represents the number of qubits that are required to run an algorithm. It also takes into account ancillary qubits. The main purpose of ancillary qubits is to help perform computations – they do not contain any input or output values. *Width* affects the simplicity of implementation: the fewer the qubits, the easier it is to implement a quantum algorithm, due to the limited amount of qubits in existing quantum computers.
- *Depth* is the metric that describes the number of quantum gates in sequence. This value affects the quantum decoherence resistance as well as the run time of an algorithm. The fewer sequenced steps

(gates), the faster and more robust the quantum circuit's performance. *Depth* is often used to describe quantum algorithms interchangeably with *width*, because it is common for one to be reduced at the expense of another.

- *Gate complexity* represents the quantity of operations (quantum gates) required to run the algorithm. Its value has an impact on run time as well as on the quantum decoherence resistance. The lower the *gate complexity*, the less susceptible the algorithm's quantum effects are to destruction by external noise.

Shor's algorithm [3] factorizes an n -bit integer using gate complexity of order $\tilde{O}(n^2)$. But, as previously described, large gate complexity comes with a large decoherence and in the result the collapse of quantum effects. Regev's algorithm [7] addresses this issue and proposes a solution with gate complexity of $\tilde{O}(n^{3/2})$. It should be pointed out that a quantum circuit of this size needs to be run $\sqrt{n}$ times. Moreover, the gate complexity can be reduced at the expense of the complexity of the classical part of the algorithm. This is the case if superpolynomial-time classical post-processing is allowed. In that case, for $1 < \epsilon \leq 1/2$, the gate complexity is of order $\tilde{O}(n^{3/2-\epsilon})$ with the classical post-processing part (solving the hard lattice problem) of order $\tilde{O}(e^{n^{2\epsilon}})$. Under these circumstances, the number of times that the quantum circuit needs to be run is $n^{1/2+\epsilon}$. Regev states that the lower gate complexity comes at the expense of width, because his algorithm has a width of order $O(n^{3/2})$, while the optimized Shor's algorithm $O(n)$. If it comes to depth, Shor's algorithm can be implemented with $\tilde{O}(n^3)$ [15], $\tilde{O}(n)$ [16] or even $\tilde{O}(\log n)$ [17] depending on the optimization and width of the algorithm. Currently, an optimized Shor's algorithm uses depth of order $\tilde{O}(n)$. Nevertheless, Regev states that his algorithm depth is smaller by $\tilde{O}(n^{1/2})$ than the original Shor's algorithm.

Paper [9] provides a theoretical and mathematical overview of the improvements and modifications to the implementation of Regev's algorithm. Based on this work, Table 1 presents a comparison of Shor's and Regev's algorithms in terms of theoretical circuit width and gate complexity for various multiplication algorithms. The comparison concerns just one run of the quantum algorithm: Shor's algorithm needs only $o(1)$ runs, while Regev's algorithm needs $o(\sqrt{n})$ runs in the proposed configuration.

In our work, we use $d \in \{\lceil \sqrt{n} \rceil, \lfloor \sqrt{n} \rfloor\}$ quantum input registers, each of them having width $qd \in \{\lceil \frac{n}{d} + d \rceil, \lfloor \frac{n}{d} + d \rfloor\}$. Note that qd is a symbolic parameter name and should not be interpreted as a multiplication. Our output register has width n , while an ancilla register uses $n + 1$ qubits. Therefore, the upper bound width of our algorithm is calculated as:

$$O(d \cdot qd + n + n + 1) = O(\sqrt{n} \cdot 2\sqrt{n} + 2n + 1) = O(4n + 1) = O(n).$$

As mentioned previously, Regev's algorithm is a multidimensional version of Shor's algorithm. The gate with the highest complexity metric upper bound value in both algorithms is a modulo exponentiation gate. In Shor's quantum circuit, there is only one modulo exponentiation gate, while in Regev's circuit, there are d modulo exponentiation gates, but each of them uses a smaller number of qubits. Regardless of the algorithm, a modulo exponentiation gate consists of $2w$ multiplications and each multiplication consists of w addition operations, where w is the width of the input quantum register. For addition, we used Häner implementation [11,14], which has gate complexity of $O(w \log w)$. Here, w denotes the widths of the quantum input and output registers. In Regev's algorithm, the input and output registers used by the modulo exponentiation gate have different widths, and because of this, we experienced difficulties in applying theoretical Häner's complexity in our case. Thus, we decided to take into consideration the worst possible scenario and its complexity is of the order $O(w_{\text{output}} \log w_{\text{output}})$.

As a result, if it comes to gate complexity in our implementation of Regev's algorithm, it consists of d modulo exponentiation gates, each with $2qd$ multiplications, each with qd operations of addition, each of complexity $O(n \log n)$. Therefore, the upper bound gate complexity of our algorithm is calculated as follows:

$$\begin{aligned} O(d \cdot 2qd \cdot qd \cdot O(n \log n)) &= O(\sqrt{n} \cdot 2(2\sqrt{n}) \cdot 2\sqrt{n} \cdot O(n \log n)) \\ &= O(8n^{3/2} \cdot O(n \log n)) = O(n^{3/2} \cdot O(n \log n)) = O(n^{5/2} \log n). \end{aligned}$$

Table 1. Shor and Regev algorithms metrics comparison. The content is based on paper [9].

Algorithm	Multiplication algorithm	Width	Gate complexity
Shor [3]	Harvey, Hoeven [18]	$O(n \log n)$	$O(n^2 \log n)$
Shor	Schoolbook	$O(n)$	$O(n^3)$
Shor	Gidney [19], KMY [20]	$O(n)$	$O_\epsilon(n^{2+\epsilon})$
Optimized Shor [12,14,15,21,22]	Schoolbook	$(1.5 + o(1))n$	$\tilde{O}(n^3)$
Regev [7]	Schoolbook	$O(n^{3/2})$	$\tilde{O}(n^{3/2})$
Regev [7]	Harvey, Hoeven [18]	$O(n^{3/2})$	$O(n^{3/2} \log n)$
Optimized Regev [9]	Harvey, Hoeven [18]	$O(n \log n)$	$O(n^{3/2} \log n)$
Optimized Regev [9]	Gidney [19], KMY [20]	$(10.32 + o(1))n$	$\tilde{O}_\epsilon(n^{3/2+\epsilon})$
Optimized Regev [9]	Schoolbook [23]	$(10.32 + o(1))n$	$O(n^{5/2} \log n)$
Our implementation	Schoolbook	$O(n)$	$O(n^{5/2} \log n)$

The classical part of Shor’s algorithm has computational complexity $O(n)$, where $n = \lceil \log_2 N \rceil$. In the case of Regev’s algorithm, it is equal to the complexity of running Lenstra–Lenstra–Lovász (LLL) algorithm [24]. The LLL algorithm, for a given lattice $\mathcal{L}$ defined by the basis $F = \{f_1, \dots, f_z\}$, where $z \leq n$ and $f_1, \dots, f_z \in \mathbb{R}^n$, finds a set of nearly shortest vectors. The computational complexity of the LLL algorithm for such a lattice $\mathcal{L}$ is $O(z^5 n \log^3 G)$, where $G = \max(\|f_1\|_2, \dots, \|f_z\|_2)$.

In our case, the LLL algorithm is applied to the columns of the matrix B . The columns of B form a basis of a lattice in $\mathbb{R}^{d+m}$. Parameter M is the maximum Euclidean norm of the vectors in the lattice B . The details of lattice B are provided in Section 3.2. Therefore, the computational complexity of the classical part of Regev’s algorithm is polynomial and equal to $O((d+m)^6 \log^3 M)$.

3. Regev’s Algorithm Implementation

Apart from the complexity of the quantum and classical parts, the algorithm’s performance is also affected by the hardware specifications, tools, chosen libraries, and existing implementations used. For developing and running the quantum part of the algorithm, we used the open-source software development kit created by IBM Research, that is, Qiskit. We used three libraries provided by this SDK. The first was `qiskit` (version 1.2.4), which is used to create quantum circuits and to manage the classical–quantum data flow. The second library was `qiskit-aer` (version 0.15.1), which comes with a simulator that simulates quantum states on a classical computer, so in practice it is a quantum computer simulator. As for the third library, `quantum-ibm-runtime` (version 0.32.0) provides the possibility of running the developed quantum circuit on IBM cloud quantum computers.

The authors of this paper implemented Regev’s algorithm based on Stępień’s publicly available implementation of Shor’s algorithm [11]. Stępień’s work focuses on the analysis and implementation of different variants of Shor’s algorithm, with particular emphasis on the construction and optimization of quantum circuits for modular exponentiation. His study includes a comparative evaluation of modular exponentiation gates proposed by Beauregard [12], Takahashi [13], Häner [14], analyzing their circuit width, depth, and asymptotic complexity, as well as proposing an original hybrid approach that achieves favorable trade-offs between these parameters. The modular exponentiation gates mentioned above differ mainly in the approach to constructing the addition gate.

The considered modular exponentiation gates differ primarily in their approaches to constructing the addition gate. In particular, they vary in the number of input qubits, as well as in the design of the constant-addition gate and the comparator modules.

The source code used as a basis for our work is available at <https://github.com/bartek-bartlomiej/master-thesis> (accessed on March 17, 2026). As mentioned in Section 1, our quantum circuit implementation builds directly on Stępień’s results, and in particular it adopts his variant of the Häner modular exponentiation gate, which was identified as offering an advantageous balance between circuit size and depth. The quantum circuit is described in detail in Section 3.1.2.

As for the classical part, in order to perform computation on lattices, we used a NumPy library. It offers fast computation on matrices and vectors [25]. Unfortunately, that library does not provide an implementation of the LLL algorithm. Therefore, we used an `oll11` package in version 1.0.2, which is a dedicated Python tool providing an implementation of the LLL algorithm.

3.1. Quantum Part

The quantum part was developed using the Qiskit SDK. In addition, we used a quantum computer simulator provided with the `qiskit-aer` package. It performs calculations on a *statevector* that stores the probabilities of the basic states of the qubits. Using this tool, the Qiskit SDK runs a quantum circuit once and computes the full pre-measurement state. Then an arbitrary number of measurements can be simulated. In a real quantum computer, the pre-measurement quantum state is not accessible; therefore, the quantum circuit must be executed separately for each measurement outcome. This approach in the simulator does not affect the accuracy of the results, since it directly follows the rules of quantum mechanics in the Qiskit framework. As mentioned in Section 1, we performed the simulations under ideal conditions, neglecting the effects of decoherence.

This feature is particularly beneficial in the context of Regev’s algorithm. The quantum part theoretically requires $d + 4$ executions. Using a quantum computer simulator, we simulate this behavior by performing a single execution of the quantum part, collecting 128 measurements, and subsequently selecting $d + 4$ vectors from the results rather than running a quantum part multiple times. The selection process of these $d + 4$ vectors is performed using one of the three methods described above in Section 3.2. This approach significantly reduced the overall runtime of the algorithm, enabling a detailed investigation into the impact of various parameters of Regev’s algorithm. In a real-world scenario, executing Regev’s algorithm on a physical quantum computer could further optimize the runtime of the quantum component. Even if $d + 4$ quantum part executions are required, this approach may still outperform the single-run simulation employed in this study.

Additionally, `qiskit-aer` package comes with different simulation methods, each with distinct properties. For example, some of them can support computation on a GPU or can simulate noise in quantum circuits. They also differ in the number of qubits they can simulate (excluding classical registers used for measurements) and in their hardware requirements. It is also possible to introduce noise to the simulator. In our implementation, we used the *statevector* simulator method, as it allows for simulating ideal circuit (without noise simulation) with the usage of CPU. However, this approach comes at the cost of large RAM consumption [26]. A *statevector* of n -qubits uses 2^n complex values, each requiring 16 bytes of memory. This means that a 32-qubit vector uses 64 GB of RAM. Our environment enabled us to perform computations on up to 29 qubits, which requires 8 GB of RAM.

The applied quantum simulation method did not simulate quantum noise. In case of some of other simulation methods, or real quantum computers, it is necessary to apply quantum error mitigation techniques [27]. Qiskit also provides error mitigation techniques for running a quantum circuit on real quantum computers with its `qiskit-ibm-runtime` package [28].

3.1.1. Parameters

Regev’s algorithm is a multidimensional version of Shor’s algorithm. In our work, the number of dimensions is equal to the value of the parameter d . Regev recommends its value to be $\lceil \sqrt{n} \rceil$ for the n -bit integer N . We decided to analyze two possible values for d and those are $\lceil \sqrt{n} \rceil$ (later referenced as “`ceil`”) and $\lfloor \sqrt{n} \rfloor$ (later referenced as “`floor`”), so in the result $d \in \{\lceil \sqrt{n} \rceil, \lfloor \sqrt{n} \rfloor\}$. In practice, the discussed parameter denotes the number of quantum input registers and the number of squared primes. This in turn has a direct impact on the number of modulo exponentiation gates as well as the number of quantum Fourier transform gates and thereby on the quantum circuit width, depth and gate complexity. The complexity of the algorithm was discussed in Section 2.2.

In addition to the parameter d , we put in place the parameter qd , which defines width of each of the quantum input register. It can also take two possible values that result in $qd \in \{\lceil \frac{n}{d} + d \rceil, \lfloor \frac{n}{d} + d \rfloor\}$. Later, a version using the `ceil` function is referenced as “`ceil`” and the `floor` version as “`floor`”. Thus, for example, the combination of the d `ceil` version with the qd `floor` version might be called `ceil_floor`. Regev’s recommendation for the parameter qd is $\lceil \frac{n}{d} + d \rceil$ for an n -bit integer N . This parameter denotes the upper bound of the exponents $(z_1, z_2, \dots, z_d)$ in the process of creating a superposition of a function. In case of a quantum circuit, the parameter qd has an impact on the input register widths, and the widths of the quantum input registers affect the overall width, depth and gate complexity.

Due to the limitations of the quantum computer simulator and hardware specifications, there is a limited value of N that we were able to factorize for `ceil_ceil`, `ceil_floor` parameters combinations. These limitations also affect Shor’s algorithm, which runs for $N = 77$ at the maximum for our simulations runs. Table 2 illustrates the largest factorized number depending on the parameters d and qd .

Table 2. The biggest factorized number N depending on d and qd parameters.

d	qd	N
ceil	ceil	57
ceil	floor	57
floor	ceil	119
floor	floor	143

3.1.2. Regev’s Quantum Circuit

The implementation initializes a uniform superposition over the d quantum input registers using Hadamard gates, which constitutes the first step in the quantum component of the algorithm. In Regev’s paper [29], a Gaussian distribution is proposed instead. However, Kiebert notes that “It is possible that the algorithm still works if one takes a uniform superposition restricted to a grid, but this would lead to less convenient quantum Fourier transform calculations” [8]. Due to time constraints and the complexity of implementing a Gaussian distribution, we chose to use a uniform distribution instead. Consequently, our implementation represents a modified variant of Regev’s algorithm rather than a full implementation.

A Pauli-X gate is applied to the first qubit of the output register, which is equivalent to a NOT gate in classical computing. This operation sets the initial value of the output register to 1, enabling it to store the result of the multiplication process in subsequent steps (by default, the output register holds a value of 0). The next stage involves applying modulo exponentiation gates to each input register and the output register.

To implement this step, we used Stepień’s adaptation of Häner’s modular exponentiation gate. This variant was selected because, among the four variants analyzed in Stepień’s work [11], it yields a circuit with the best asymptotic complexity in terms of both size and depth. Regev’s algorithm employs d quantum input registers of smaller width qd , in contrast to Shor’s algorithm, which uses a single quantum input register of width n . Consequently, we modified Häner’s modulo exponentiation gate by eliminating redundant quantum input qubits in each of the d gates. In addition, we adopted Stepień’s naming convention [11] for quantum gates. The modulo exponentiation gate is referred to as the “Exp” gate and is decomposed into qd multiplication gates, denoted as “C-U” gates. Following this, the output register is ignored (measured), as its value is not relevant to our objectives. The next step involves applying Quantum Fourier Transform (QFT) gates to each quantum input register. Finally, the values of the quantum input registers are measured and stored in the classical registers.

To illustrate the impact of this decomposition, we present the quantum circuits in two forms: general and decomposed. This approach emphasizes that any modification leading to the addition of a single “Exp” quantum gate in the circuit actually results in an increase in the number of underlying quantum gates that make up that additional gate. Figures 1 and 2 depict a general and decomposed quantum circuits accordingly. The factorized number is $57 = 3 \cdot 19$ and a parameter combination is `ceil_ceil` (parameters d and qd are taken in the `ceil` version, which results in $d = 3$ and $qd = 5$). It is worth mentioning that in reality, the “Exp” gates are small quantum circuits. This also holds for the “C-U” gates, which are composed of the modulo adder gates. These, in turn, are composed of gates such as adder or carry, and these gates are finally composed from primary gates such as Pauli-X (X) or Hadamard gates. These gates compositions are described in detail in Stepień’s work [11].

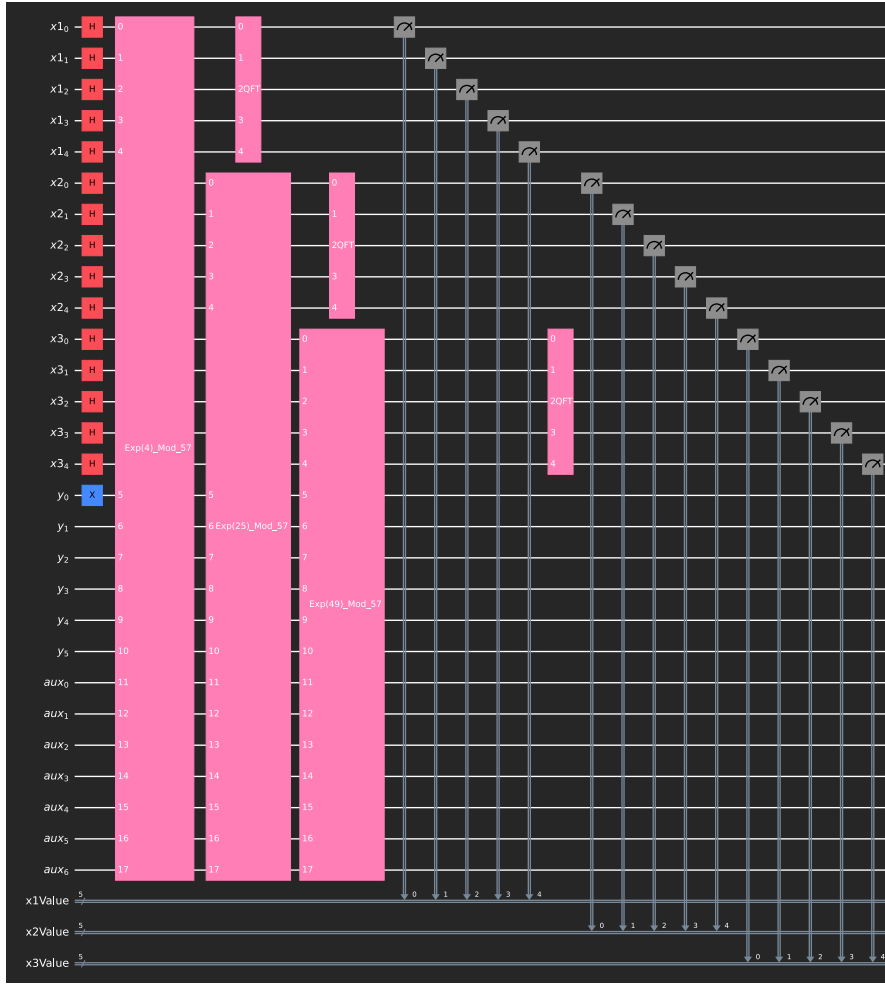


Figure 1. General quantum circuit for $N = 57$, d ceil and qd ceil (ceil_ceil) parameters.

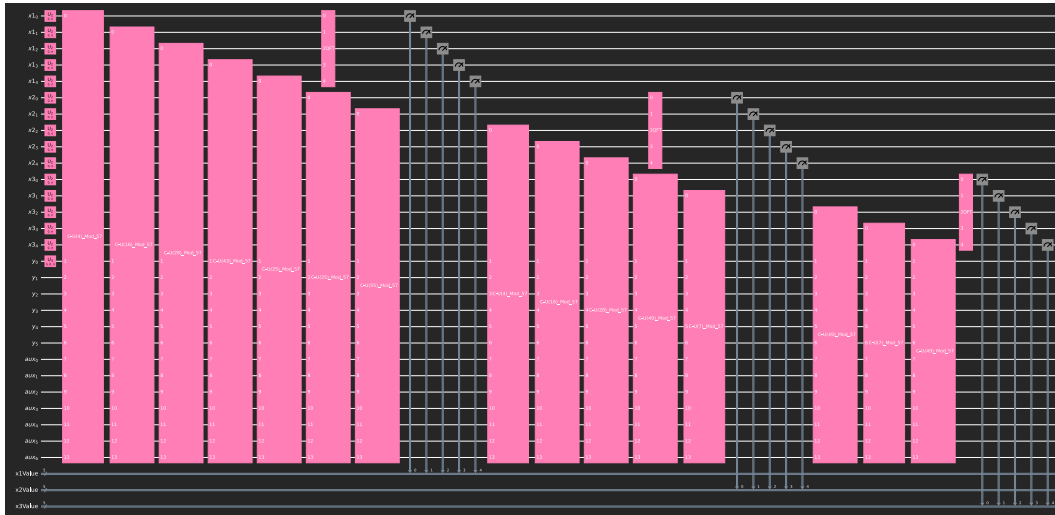


Figure 2. Decomposed quantum circuit for $N = 57$, d ceil and qd ceil (ceil_ceil) parameters.

3.1.3. Output Vectors

Applying the Quantum Fourier Transform (QFT) and measuring the states of each quantum input register yields a specific output vector. For example, the output vector $[10, 22, 21]$ consists of three values, each corresponding to the measurement of a quantum input register at the end of the quantum circuit. Measurement of the first quantum input register produces the value 10, the second register yields 22, and the third register results in 21.

In Qiskit's nomenclature, measurements are referred to as *shots*. Due to the probabilistic nature of qubits in a quantum superposition, the values in the output vector may vary across multiple runs of the quantum circuit. Another consequence of qubit probabilistic behavior is that certain vectors have a higher probability of occurrence than others. As a result, repeated measurements of the quantum circuit output reveal variations in the frequency of observed vectors.

In summary, for a fixed N and parameter set, the measured vectors and their frequencies are governed by the underlying probability distribution and may vary between measurement collections.

In our implementation, we collected 128 measurements for each N using a single quantum circuit run. Consequently, the total number of vector occurrences sums up to 128. For example, the difference between Figures 3 and 4 illustrates the impact of quantum probabilities on the occurrences of the output vectors for $N = 51$ with the `ceil_ceil` and `ceil_floor` parameter settings.

```
vector with 9 shots: [20, 12, 8]
vector with 11 shots: [24, 8, 16]
vector with 13 shots: [4, 28, 8]
vector with 16 shots: [16, 16, 0]
vector with 17 shots: [0, 0, 0]
vector with 20 shots: [8, 24, 16]
vector with 21 shots: [12, 20, 24]
vector with 21 shots: [28, 4, 24]
```

Figure 3. Output vector measurements for $N = 51$ and `ceil_ceil` parameters.

```
vector with 13 shots: [28, 4, 24]
vector with 14 shots: [16, 16, 0]
vector with 15 shots: [12, 20, 24]
vector with 15 shots: [0, 0, 0]
vector with 16 shots: [4, 28, 8]
vector with 17 shots: [8, 24, 16]
vector with 19 shots: [24, 8, 16]
vector with 19 shots: [20, 12, 8]
```

Figure 4. Output vector measurements for $N = 51$ and `ceil_floor` parameters.

3.2. Classical Part

The goal of the classical part of Regev's algorithm is to process the output of a quantum circuit to factorize a given number N . The environment for computations for the classical part was the same as for the quantum part. The used CPU model name is Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz. It has 12 logical cores (6 cores with 2 logical threads per core) and operates on x86_64 architecture in 64-bit mode. Its minimum clock speed is 800 MHz while maximum (with the Turbo Boost mechanism) is 4100 MHz. RAM has the total size of 16 Gi with the speed of 2667 MT/s (megatransfers per second).

To explain the classical part of Regev's algorithm, it is necessary to introduce some new terms. Let us define the lattice $\mathcal{L}$, which represents trivial (that is, -1 or 1) and non-trivial square roots of unity modulo- N :

$$\begin{aligned}\mathcal{L} &= \left\{ (z_1, \dots, z_n) \in \mathbb{Z}^d \mid \left(\prod_i b_i^{z_i} \right)^2 = 1 \bmod N \right\} \\ &= \left\{ (z_1, \dots, z_n) \in \mathbb{Z}^d \mid \prod_i a_i^{z_i} = 1 \bmod N \right\} \subset \mathbb{Z}^d,\end{aligned}$$

(note that $b = (b_1, \dots, b_d)$ and $a = (b_1^2, \dots, b_d^2)$ were defined in Section 2.1).

Let us define the sublattice $\mathcal{L}_0$ of lattice $\mathcal{L}$, which represents trivial square roots of unity modulo- N :

$$\mathcal{L}_0 = \left\{ (z_1, \dots, z_n) \in \mathbb{Z}^d \mid \prod_i b_i^{z_i} \in \{-1, 1\} \bmod N \right\} \subseteq \mathcal{L}.$$

It is worth noting that the lattice $\mathcal{L} \setminus \mathcal{L}_0$ represents only not-trivial square roots of unity modulo N , which are the desired elements to factorize N with the use of Regev's algorithm.

In our implementation, we used the parameters proposed in Kiebert's work [8], which are defined as follows:

$$m = d + 4;$$

$$R = 6T\sqrt{(d+5)(2d+4)(d/2)} \cdot 2^{(n+1)/(d+4)+d+2},$$

where T is the upper bound on the norm of the smallest vector in $\mathcal{L} \setminus \mathcal{L}_0$;

$$t = 1 + \lceil \log_2(\sqrt{d}R) \rceil;$$

$$\delta = \frac{\sqrt{d}}{\sqrt{2}R};$$

$$S = \delta^{-1}.$$

The matrix B is defined as follows:

$$B = \left(\begin{array}{c|c} I_{d \times d} & 0 \\ \hline S \cdot \omega_0 & \\ \vdots & S \cdot I_{m \times m} \\ S \cdot \omega_m & \end{array} \right),$$

where $\{\omega_0, \dots, \omega_n\}$ are the vectors obtained from the quantum circuit output, divided by 2^t .

However, we rounded the parameter S to the nearest integer. Without this adjustment, the LLL algorithm for some numbers could reduce the vectors to a zero vector. For small numbers, it is possible to find exactly such a vector with complexity $O(qd^d)$, but it is equivalent to finding a result vector that allows computing p and q for N without a usage lattice and LLL algorithm. For large numbers, this approach is very time consuming. To make our solution scalable, we approximate the value of T using a heuristic assumption introduced in Regev's work. It states that there exists a vector in $\mathcal{L} \setminus \mathcal{L}_0$ with the norm at most $T = \exp(O(\frac{n}{d}))$. This assumption provides an estimate for the likely norm of the desired vectors before applying the LLL algorithm. Based on this, we can predict the value of R , which influences the determinant of the matrix B . By estimating R , we can adjust the determinant of B to ensure that, after applying the LLL algorithm, we are likely to obtain vectors with the desired norm. The correlation between the determinant of matrix B and the impact on the length of vectors after applying the LLL algorithm, along with a mathematical proof, is detailed in Kiebert's work [24]. For small N ($N \leq 57$), we found that a good approximation for T is $\lceil \exp(\frac{n}{2d}) \rceil$. This result was obtained by calculating accurate values of T for a few cases and determining the best-fitting curve. The comparison between the exact values of T and the approximation using the function $\exp(\frac{n}{2d})$ is presented in Table 3.

Table 3. Comparison of exact values of T and their approximations using function $\exp(\frac{n}{2d})$ for selected values of N .

N	15	21	35	51	57
value of n	4	5	6	6	6
value of d	2	3	3	3	3
exact value of T	3	2	3	3	3
approximate value of T	2.718	2.301	2.718	2.718	2.718

Following the implementation of the algorithm and its execution on a simulated quantum computer under ideal conditions, we conducted a series of experiments. Selecting multiple output vectors from the quantum circuit consists

in executing the circuit once and then performing multiple measurements of the *statevector*, rather than running the quantum circuit independently multiple times. Therefore, we collect 128 vectors for every parameter (`ceil_ceil`, `ceil_floor`, `floor_ceil` and `floor_floor`) and for all values N that our hardware allows us to simulate (as indicated in Table 2). Therefore, after measuring quantum output vectors, we simulate many independent runs of circuits. In our program, we implemented a method called `run_file_data_analyzer`, which analyzes the outputs of quantum circuit saved in files. The purpose of this function is to measure the effectiveness of finding square roots of unity modulo- N (highlighting non-trivial ones) for our implementation of Regev’s algorithm. This method takes four parameters. Two of them specify which quantum output files to analyze, while the other two define the type of test (parameter `type_of_test`) and the number of repetitions (parameter `number_of_combination`). We implemented three different types of tests.

In the first test, we randomly took $d + 4$ vectors according to the probability of their return by the quantum circuit. This approach allows us to simulate $d + 4$ executions of the quantum circuit.

In the second test, we wanted to check whether there exist some vectors that, despite frequent occurrence among quantum circuit output vectors, might not be a good approximation of the vectors from the dual lattice $\mathcal{L}^*$. To achieve that, we selected each vector from the quantum circuit’s output with equal probability to form a set of $d + 4$ vectors, regardless of how many times a specific vector was returned. If the results of the second test were better than those of the first test, it would indicate that certain frequently returned vectors – such as $[0, 0, 0]$ for $N = 21$ – reduce the algorithm’s effectiveness.

In the third test, we generated completely random vectors. The length of the vector is defined by parameter d . In the quantum circuit, each coordinate of the vector is measured from qd qubits. That means that if it was unknown what happens in the circuit, we could assume that the measurement would return values from the range $[0, 2^{qd})$. Thus, according to this method, we create vectors of length d with random coordinates in the range $[0, 2^{qd})$. This test allows us to verify whether the correct factoring of N in our implementation of Regev’s algorithm is not a coincidence.

In each test, we measured two parameters. The first parameter is a percentage of vectors that, after computation, returns the square root of unity modulo- N . The second parameter is the percentage of vectors that return non-trivial square roots of unity modulo- N , i.e., values not equal to -1 or 1 . For every number N and parameters d and qd that we tried to factorize on the quantum circuit simulator, we ran the three tests 1000 times to gather sufficient statistics.

3.3. Limitations

As discussed in the preceding sections, our implementation is subject to several limitations. Due to memory constraints, computations were restricted to circuits of at most 29 qubits, which limited the problem size to relatively small instances, with the largest factorized number being $N = 143$. All simulations were performed under ideal, noise-free conditions; consequently, the algorithm’s effectiveness is expected to be lower in practical, noisy settings. Furthermore, the use of a uniform superposition in place of the Gaussian superposition prescribed in Regev’s algorithm may have additionally affected the observed performance.

4. Comparative Analysis

Following the implementation of the algorithm and its execution on a simulated quantum computer under ideal conditions, we conducted a series of experiments. The research focused on the runtime and effectiveness depending on the various combinations of parameters. We also measured the runtime and effectiveness of Shor’s algorithm in the version implemented by Śtepień [11] and compared those results with the performance of Regev’s algorithm. Effectiveness was defined as the factorization success rate, meaning the proportion of instances in which a number was successfully factored. The set of numbers below presents all the factorized numbers N :

$$15, 21, 33, 35, 39, 51, 55, 57, 65, 69, 77, 85, 91, 95, 119, 143. \quad (2)$$

It is important to note that, as discussed in Section 1, Shor’s algorithm has benefited from over two decades of research and optimization, whereas Regev’s algorithm is a relatively recent development, having been proposed only about three years ago.

4.1. Runtime Analysis

4.1.1. Methodology

This section presents the Regev’s algorithm quantum and classical part runtime performance for all analyzed parameters configurations and their comparison with the Shor’s algorithms runtime. We used a Qiskit SDK with a

quantum computer simulator. To collect runtime data, the quantum and classical parts of the algorithms were executed only once. This approach was chosen to focus on the impact of parameter variations and due to the time constraints. In practice, the runtime of a quantum part on a real quantum computer should be considerably faster.

In Section 3.1.1, we mentioned that for each of the parameters combination there was a maximum factorized number N (Table 2). This is the reason why the data series on the following graphs have different maximum values of N on the X-axis. Also, we decided to analyze the quantum and classical parts separately because the runtime of the classical parts in Regev's and Shor's algorithms is negligibly small (the classical part runs in about half a second, while the quantum part runs for minutes to hours). Thus, the comparison of the algorithms' speeds corresponds to the comparison between their quantum parts.

4.1.2. Results

Figure 5 presents the runtime of the Regev's algorithm for d parameter in `ceil` version along with the performance of Shor's algorithm. For an analyzed N range, Shor's algorithm is considerably faster, which aligns with Regev's remark that his algorithm is asymptotically faster than Shor's algorithm. This means that there exists a value of N for which Regev's algorithm outperforms Shor's algorithm in terms of speed, although this value might be very large. It can be seen that for the number $N = 33$ runtime for all presented runs increased significantly. This is because the previous number $N = 21$ is an integer of $n = 5$ bits, while $N = 33$ is an integer of $n = 6$. The number that is 1 bit larger affected quantum output and ancillary registers by adding one qubit, so it increased the width of a quantum circuit by two qubits and resulted in an increase in the runtime of the quantum part of the algorithm.

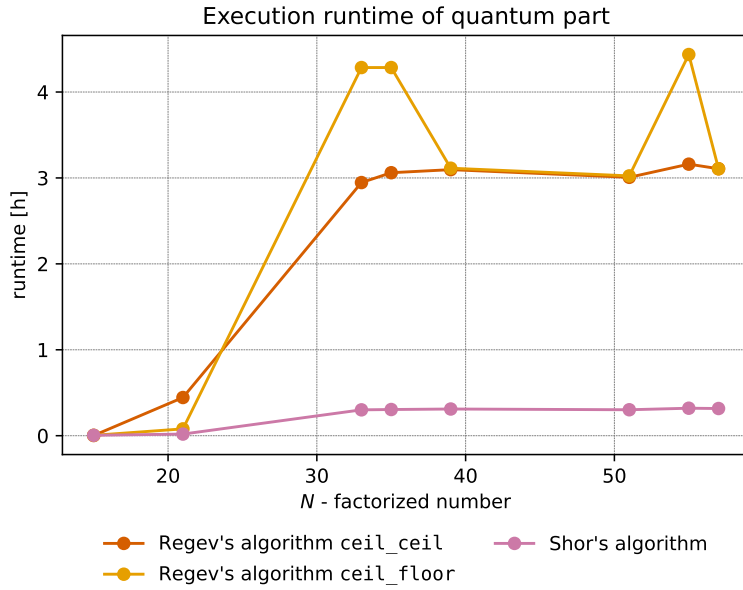


Figure 5. Runtime comparison for `ceil_ceil` and `ceil_floor` parameters.

Figure 6 depicts a comparison between Regev's algorithm with d parameter in `floor` version and Shor's algorithm. Contrary to the previous graph, here the maximum N is equal to 143 for `floor_floor` parameters. In the case of Shor's algorithm, the maximum factorized number was 77. In the same way as before, there is a significant increase in run-time for number $N = 65$ regarding all of the runs. Similarly, $N = 65$ is 1 bit larger than previously analyzed number $N = 57$. An analogous situation can be observed in the case of the number $N = 143$ for `floor_floor` parameters. For this set of parameters, Shor's algorithm is slower, but this is at the cost of the effectiveness of the Regev's algorithm.

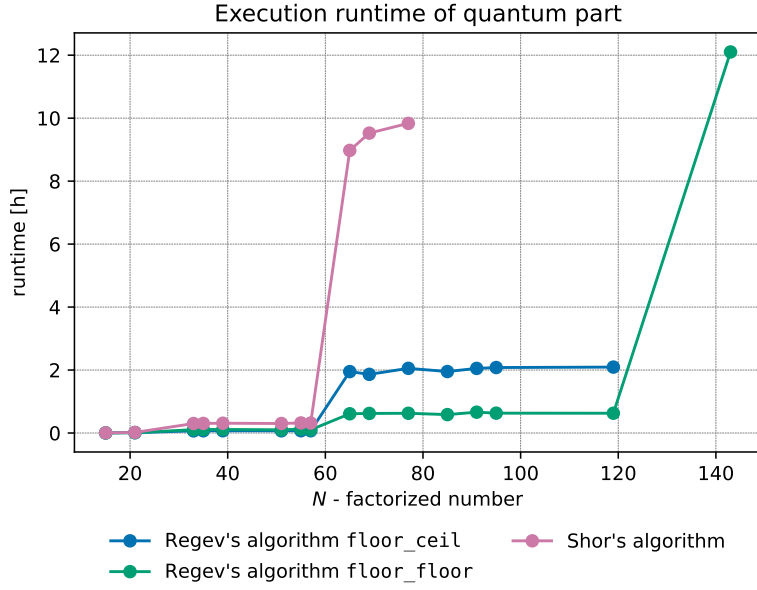


Figure 6. Quantum part runtime comparison for `floor_ceil` and `floor_floor` parameters.

For the ranges and parameters of the analyzed number N and the parameters, in terms of speed, the best performance has parameters d in the `floor` and qd in `floor` modes. This set of parameters indicates the smallest number of dimensions as well as the smallest exponentiation boundary among the analyzed parameters. In the result, this set of parameters directly affects a number of quantum input registers, as well as the width of the quantum circuit. The conclusion is that under the given conditions regarding N and the parameters, fewer dimensions and smaller exponentiation boundaries result in a faster quantum part runtime of Regev's algorithm. Moreover, for this set of parameters, Regev's algorithm was able to factorize a bigger maximum number N than the other parameters. Table 4 presents a comparison of the average factorization times for runs with $N \leq 57$. The results indicate that the shortest runtimes are achieved when the parameter d is set to the `floor` mode.

Table 4. Average runtime up to $N = 57$.

Algorithm	Average runtime
Regev (<code>ceil_ceil</code>)	2 h 21 min 11.24 s
Regev (<code>ceil_floor</code>)	2 h 47 min 53.43 s
Regev (<code>floor_ceil</code>)	3 min 18.27 s
Regev (<code>floor_floor</code>)	5 min 2.21 s
Shor	14 min 3 s

The classical computations underlying Regev's algorithm exhibit greater complexity than those of Shor's algorithm. Consequently, we compared the runtimes of these classical components to evaluate how these differences might affect the overall computation time. Figure 7 illustrates the run-time of classical computations for Regev's and Shor's algorithms. It can be seen that Shor's algorithm's classical part is significantly faster than Regev's classical part, but considering a quantum part runtime, it is negligible for the whole algorithm's speed. When comparing Regev's algorithm parameters, the faster are those with d parameter in the `floor` mode. It can be observed that the qd parameter does not have a significant impact on the algorithm's classical parts' speed. Similarly to the execution time of quantum parts, the best performance among the combinations of parameters analyzed is for d in the `floor` and qd in `floor` modes.

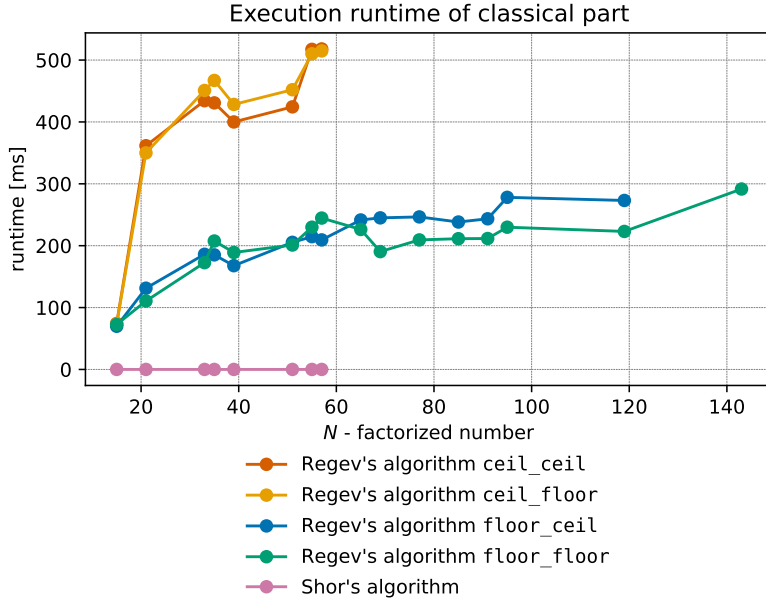


Figure 7. Classical part runtime comparison for all parameters configurations.

4.2. Effectiveness Analysis

4.2.1. Methodology

In this section, we present the results of the effectiveness of our implementation of Regev's algorithm. To automate the testing process, we implemented the function `run_file_data_analyzer`, as described in Section 3.2. We define one test run as taking $d + 4$ vectors using the method specified by the parameter `type_of_test`, constructing a matrix B from these vectors, executing the LLL algorithm, and verifying two properties. The first property checks whether there exists a vector among the returned ones that allows recovering a square root of unity modulo- N , i.e., whether the returned vector belongs to the lattice $\mathcal{L}$. The second property checks whether there exists a vector among the returned ones that allows recovering a non-trivial square root of unity modulo- N , i.e., whether the returned vector belongs to the lattice $\mathcal{L} \setminus \mathcal{L}_0$.

Firstly, we tested the effectiveness of our implementation using parameters defined in Kiebert's work [8] to create matrix B . We measured the effectiveness for all values of N , where we ran the quantum circuit, using all combinations of rounding for d and qd , i.e., `ceil_ceil`, `ceil_floor`, `floor_ceil`, and `floor_floor`. For these parameters, we performed the three types of tests, which were thoroughly described in Section 3.2. For each rounding method, we prepared the graph which presents the results for all three combined tests, allowing for easier comparison of the test outcomes.

4.2.2. Results

The results of the tests for the parameter `ceil_ceil` are shown in Figure 8. It is evident that for the first type of test, the effectiveness of finding square roots of unity modulo- N (both trivial and non-trivial) declines as the factorized number N increases. For the second type of test, the effectiveness decreases up to $N = 51$, after which it increases slightly for $N = 55$ and $N = 57$. In the third type of test, the effectiveness decreases up to $N = 55$ and then increases for $N = 57$. For the parameter `ceil_ceil`, the highest effectiveness was achieved in the third type of test, although the effectiveness for the second test was nearly identical, except for $N = 51$. The first test exhibited the lowest effectiveness. The general shape of the plots across all tests for parameter `ceil_ceil` is similar.

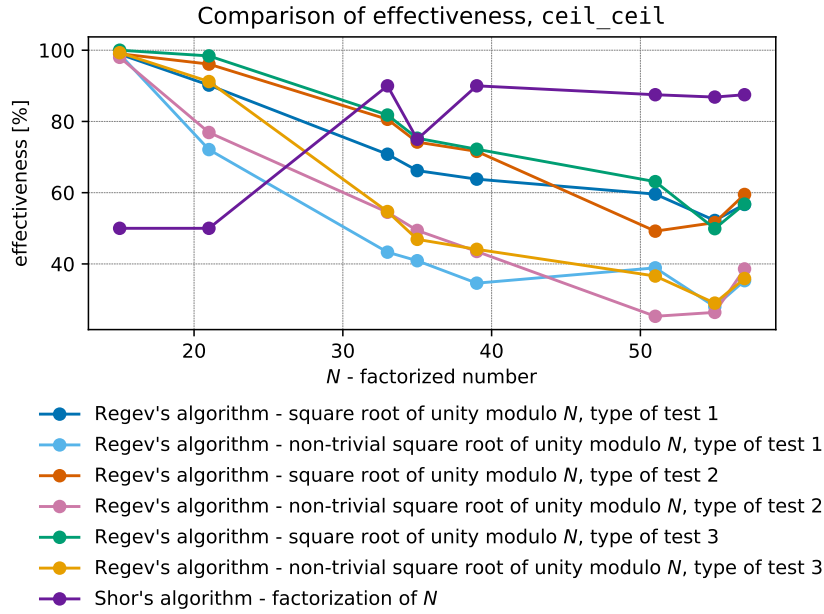


Figure 8. Effectiveness comparison for `ceil_ceil` for all types of test.

The results of the tests for the parameter `ceil_floor` are shown in Figure 9. The results are very similar to those obtained for the parameter `ceil_ceil`, with the shapes of the figures being almost identical. We observe increases and decreases for the same values of the parameter N . However, the effectiveness of all three tests shows more consistent values compared to the results for the parameter `ceil_ceil`.

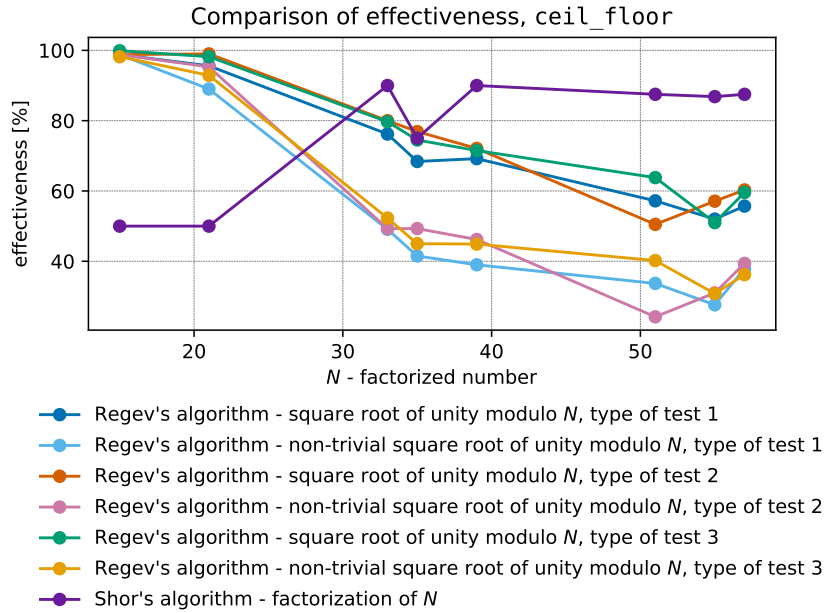


Figure 9. Effectiveness comparison for `ceil_floor` for all types of test.

For both parameters, `ceil_ceil` and `ceil_floor`, we observe that for $N = 15$ and $N = 21$, the effectiveness of factorizing N using our implementation of Regev's algorithm is better than that of Shor's algorithm. However, after $N = 33$, Shor's algorithm stabilizes at an effectiveness of approximately 88%, while Regev's algorithm exhibits a declining trend and performs significantly less efficiently than Shor's algorithm.

The results of the tests for the parameter `floor_ceil` are shown in Figure 10. All three tests have a similar shape in their figures. For the first type of test, we observe a decreasing trend, with a single peak at $N = 51$. For the

second and third types of test, a similar declining trend is evident, but the results show smaller peaks, with the third test having the smallest peaks.

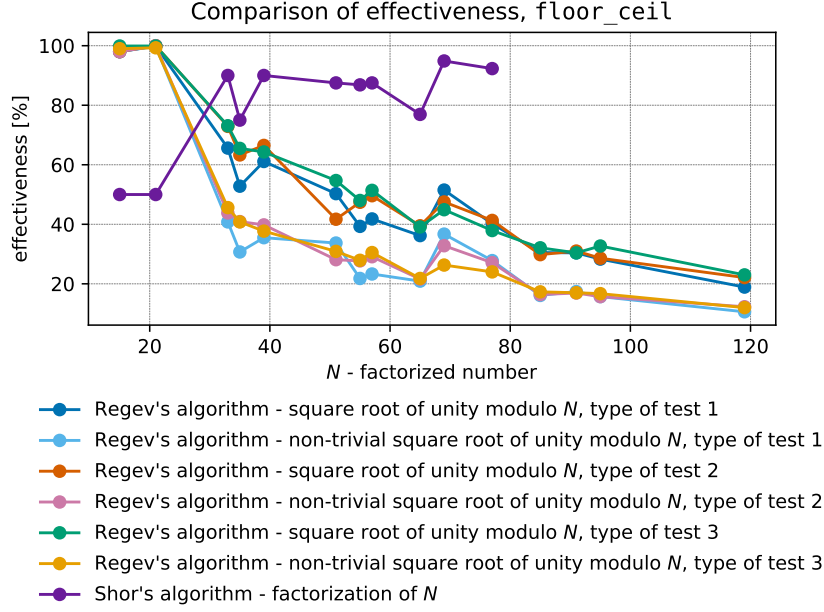


Figure 10. Effectiveness comparison for floor_ceil for all types of test.

The results of the tests for the parameter floor_floor are shown in Figure 11. For the first type of test, we observe a declining trend, with a single peak at $N = 69$. For the second and third types of tests, the shapes of the functions are very similar, with two significant peaks in effectiveness at $N = 57$ and $N = 69$. Both the second and third tests have a decreasing trend.

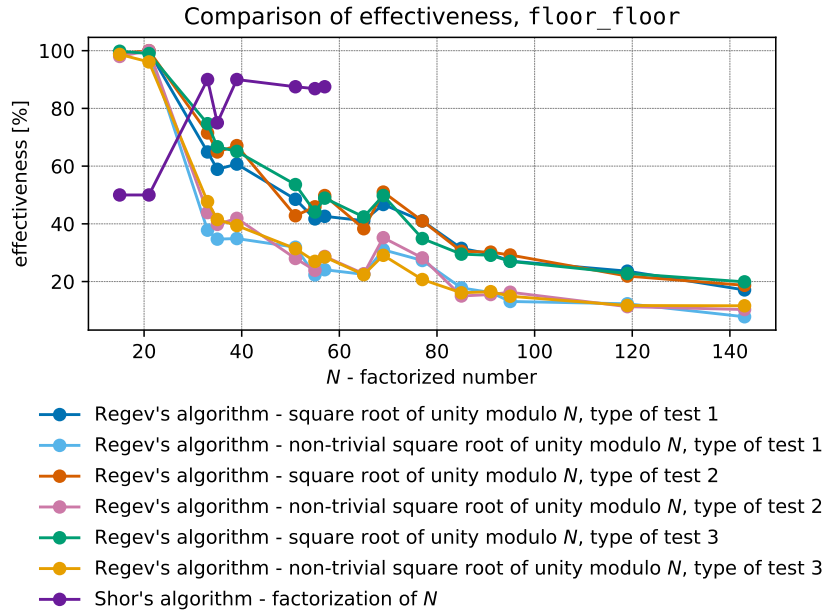


Figure 11. Effectiveness comparison for floor_floor for all types of test.

Table 5 summarizes the effectiveness of Regev's algorithm across all three test types and includes a comparison with the effectiveness of Shor's algorithm. For the parameters floor_ceil and floor_floor, we obtained worse effectiveness compared to ceil_ceil and ceil_floor. However, the choice of the rounding parameter d has a

greater impact on effectiveness than the rounding of the parameter qd . The rounding of qd does not appear to significantly influence effectiveness. Different rounding schemes for qd , when combined with the same rounding for d , result in situations where `ceil` performs better in some cases, while `floor` performs better in others. The declining trend in the effectiveness of our implementation of Regev’s algorithm as N increases is not observed in Shor’s algorithm, which stabilizes and maintains high effectiveness. This decreasing tendency in our implementation of Regev’s algorithm is consistent across all three tests and for all parameters (`ceil_ceil`, `ceil_floor`, `floor_ceil` and `floor_floor`). This consistency suggests that the parameters chosen in the classical part may not be optimal. Furthermore, we do not know the reason for the observed peaks for some values of N and declines for others.

Table 5. Average effectiveness of Regev’s algorithm variants for three test types and Shor’s algorithm.

Algorithm	Type 1	Type 2	Type 3
Regev (<code>ceil_ceil</code>)	49.00	44.94	48.34
Regev (<code>ceil_floor</code>)	52.08	47.81	55.06
Regev (<code>floor_ceil</code>)	47.95	44.20	51.46
Regev (<code>floor_floor</code>)	48.05	43.74	51.29
Shor	77.10	77.10	77.10

4.2.3. Effectiveness Analysis using t Parameter

The results presented in Section 4.2.2 indicate that our implementation, based on parameters defined in Kiebert’s work, is not efficient. Therefore, we attempted to find better parameters for the classical part. We focus our tests on the parameter t , which is used in the quantum part to create the Gaussian distribution. However, in our implementation, we utilized a uniform distribution instead of Gaussian distribution (i.e., the distribution used in the first step of the quantum circuit, before exponentiation). Thus, the parameter t was used exclusively in the classical part to transform the output vectors from the quantum circuit into approximations of vectors in the lattice $\mathcal{L}^*$.

We hypothesized that, since the parameter t is not used in the quantum part to create a Gaussian distribution, adjusting its value in the classical part might yield better results. We concentrated our research on the parameter `ceil_ceil`, as this value was proposed in Regev’s work. We conducted the first type of test for this parameter, exploring t values in the range $[2, 20]$, and attempted to identify the optimal value of t . We chose this range based on the heuristic assumption that, within these values of t , the effectiveness exhibits a significant improvement for different values of N .

Plots for several cases were presented. Figure 12 presents the results for $t = 5$. We observe that finding a non-trivial square root modulo- N for the third type of test exhibits a declining trend. However, for the first type of test, the results are not linear, and for $N = 39$, we achieved 100% effectiveness in finding a non-trivial square root modulo- N . Figure 13 shows the results for $t = 8$. For the third type of test, we again observe a declining trend. However, for the first type of test, there is a peak at $N = 51$, where we achieved 100% effectiveness in finding a non-trivial square root modulo- N . Figure 14 presents results for $t = 14$. For this parameter, the results for the first and third tests are similar, both showing a declining trend. However, there is a notable peak for the first test at $N = 51$, with significantly better effectiveness (52%) for finding a non-trivial square root modulo- N . Figure 15 presents results for $t = 2$. For this parameter, we observe that for $N = 15$ and $N = 21$, the effectiveness for the first test is much lower than for the third test. For higher values of N , the effectiveness becomes similar for both tests.

Table 6 presents, for each N , the parameter t that yielded the highest effectiveness, together with the corresponding percentage value. We achieved satisfactory results for all values of N , except for $N = 55$, where the effectiveness of finding a non-trivial square root was only 32%. The better effectiveness of factoring N than Shor’s algorithm was obtained for $N \in \{21, 33, 39, 51\}$.

Table 6. Effectiveness for different values of parameter t .

Factorized number N	21	33	35	39	51	55	57
t for which obtained the highest effectiveness	14	8	14	5	8	11	2
Effectiveness of finding square root of unity modulo- N [%]	94	86	74	100	100	62	82
Effectiveness of finding non-trivial square root of unity modulo- N [%]	74	56	48	100	98	32	70

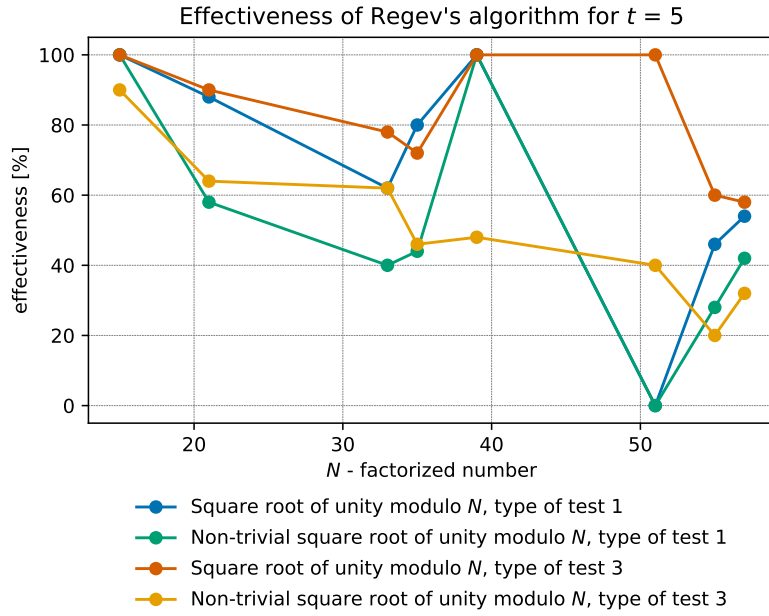


Figure 12. Effectiveness comparison for $t = 5$.

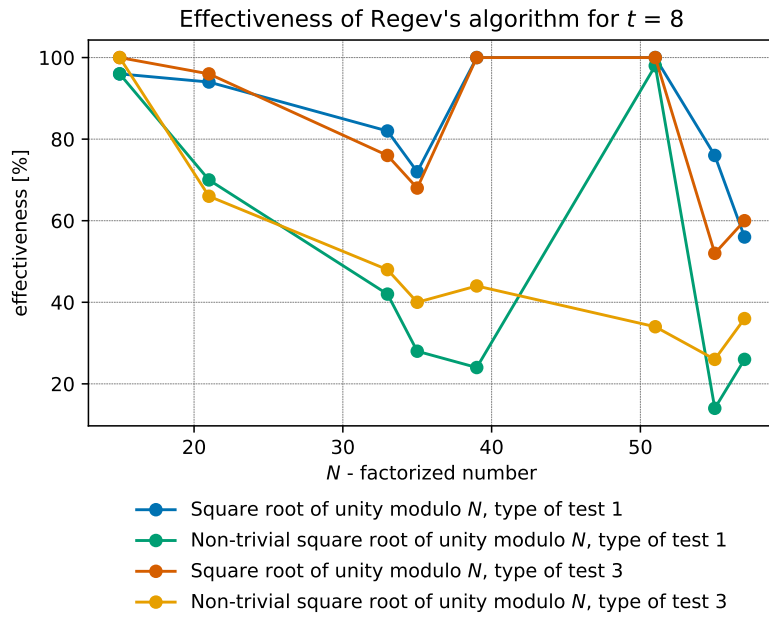


Figure 13. Effectiveness comparison for $t = 8$.

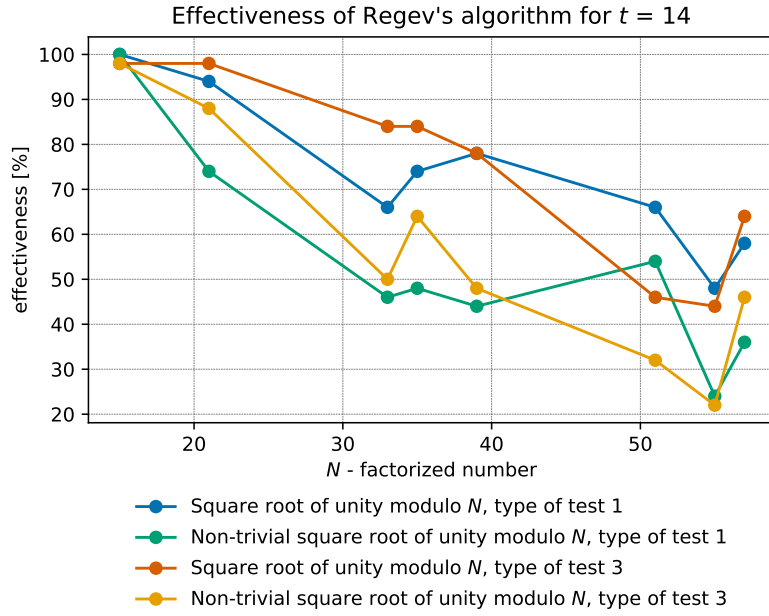


Figure 14. Effectiveness comparison for $t = 14$.

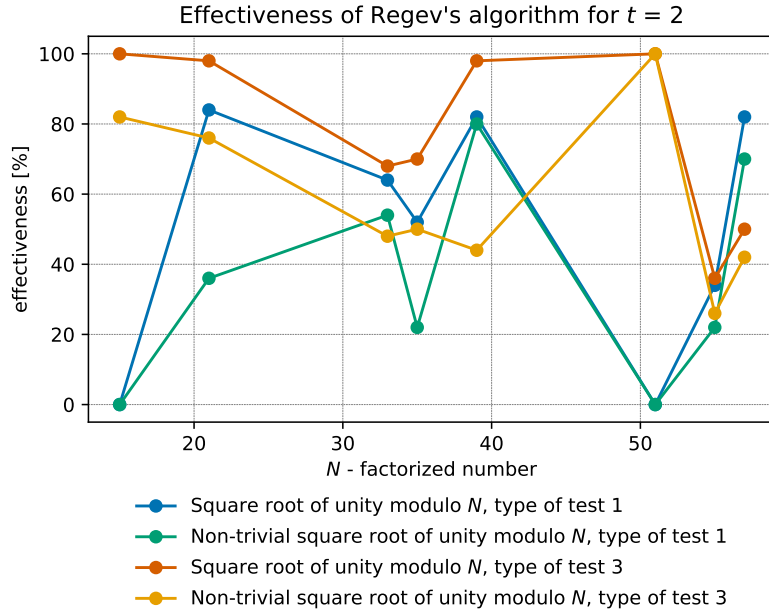


Figure 15. Effectiveness comparison for $t = 2$.

4.3. Performance Summary

In Regev's algorithm, the execution time of the quantum part depends on the chosen rounding for the parameters d and qd . This is because these parameters directly impact the circuit width (the number of qubits that need to be simulated) and gate complexity (the number of quantum gates in the quantum circuit). However, the faster the quantum part was executed, the lower the effectiveness of factoring the number N we obtained. The execution time of Regev's algorithm was significantly greater than that of Shor's algorithm (for both the quantum and classical parts). This is because Regev's algorithm is asymptotically more efficient than Shor's algorithm, and this advantage is not apparent for smaller N .

In our implementation of Regev's algorithm, the parameters proposed in Kiebert's work resulted in low effectiveness in factoring the number N . The results we obtained were similar to those obtained by using a random vector. For $N > 35$, we observed significantly lower effectiveness compared to Shor's algorithm. In our implementation of the

algorithm, due to its complexity, we did not implement the Gaussian distribution, which is correlated with parameter t . Therefore, we assumed that changing the value of t might increase the effectiveness of our implementation.

For some values of t , we achieved much greater effectiveness. For some values of N , this even improved the effectiveness over Shor's algorithm. Table 6 shows the values of t for which we achieved the highest effectiveness. Unfortunately, the correlation between the value of the parameter t and the effectiveness of finding a non-trivial square root modulo- N is not visible. However, we can conclude that the choice of parameters in the classical part has a significant impact on the algorithm's effectiveness.

5. Conclusion

This paper presents the first publicly available implementation of Regev's quantum algorithm for quantum computers. Our primary objective was to develop and analyze the performance of the algorithm, focusing on runtime and effectiveness while comparing it with Shor's algorithm. By varying parameters, we examined and visualized in graphs their influence on the execution time and effectiveness of the factorization process.

The experimental results indicate that the running time of Regev's algorithm is significantly affected by the selection of parameters d and qd , which influence both the circuit width and gate complexity. While shorter execution times led to decreased factorization effectiveness, the overall runtime of Regev's algorithm was substantially higher than that of Shor's algorithm.

Our implementation also revealed that the parameter selection proposed in Kiebert's work resulted in low effectiveness in factoring N , with results comparable to random vector selection. However, adjusting the parameter t led to improvements in effectiveness, occasionally surpassing Shor's algorithm for specific values of N . In general, our findings emphasize the importance of fine-tuning parameters to enhance the performance of Regev's algorithm.

Future work can focus on a more comprehensive implementation, including adjustments to the Gaussian distribution, which may enhance the overall speed of the algorithm. Furthermore, more research is needed to optimize Regev's algorithm for larger values of N and real quantum hardware to fully assess its potential. Leveraging supercomputers could significantly accelerate the execution time of Regev's algorithm, enabling a more extensive evaluation of its performance across various parameter settings. This would provide a definitive test environment for evaluating the true performance of Regev's algorithm.

Author Contributions

P.P., N.M. and P.C, conceptualization; P.P. and N.M., conceived and designed the experiments; P.P. and N.M., performed the experiments; P.P., N.M., M.N. and P.C, analyzed and interpreted the data; P.P. and N.M., wrote the paper; M.N. and P.C, reviewed drafts of the paper; M.N. and P.C, supervision; M.N., funding acquisition. All authors have read and agreed to the published version of the manuscript.

Funding

The research project was partly supported by the program "Excellence initiative—research university" supported by the AGH University of Krakow. This work was also supported by the EU Horizon Europe Framework Program under Grant Agreement no. 101119547 (PQ-REACT) and Grant Agreement no. 101225759 (PQ-NEXT).

Conflicts of Interest

No conflicts of interest.

Data Availability Statement

The code for the presented work, including test results, can be found in repository <https://github.com/Wlitkopa/regev-quantum-algorithm> (accessed on March 17, 2026).

Notation

In this part, a brief enumeration of the most important variables used in this paper is given.

Common Variables

N	Factorized semiprime integer, product of primes p and q .
n	Number of bits in the binary representation of number N .
p	Prime number, factor of number N .
q	Prime number, factor of number N .
$(a_1, \dots, a_d)$	The first d squared numbers that are coprime with N ; for convenience, the squares of primes are used.
$(b_1, \dots, b_d)$	The first d numbers that are coprime with N ; for convenience, the prime numbers are used.

Quantum Variables

d	Number of dimensions in Regev's algorithm, also defines the number of quantum input registers. Its value is either equal to $\lceil \sqrt{n} \rceil$ (ceil version) or equal to $\lfloor \sqrt{n} \rfloor$ (floor version).
qd	The boundary of the exponents in Regev's algorithm, also defines a width of each of the quantum input registers. Its value is either equal to $\lceil \frac{n}{d} + d \rceil$ (ceil version) or equal to $\lfloor \frac{n}{d} + d \rfloor$ (floor version).
ceil_ceil	d and qd parameters combination when d and qd are both in ceil version.
ceil_floor	d and qd parameters combination when d is in ceil version and qd is in floor version.
floor_ceil	d and qd parameters combination when d is in floor version and qd is in ceil version.
floor_floor	d and qd parameters combination when d and qd are both in floor version.

Classical Variables

$\mathcal{L}$	A lattice containing vectors that allow computing the square root of unity modulo- N .
$\mathcal{L}_0$	A lattice containing vectors that allow computing the trivial square root of unity modulo- N .
$\mathcal{L}'$	A lattice used to retrieve the period vector from the vectors obtained in the quantum part.
$\mathcal{L}^*$	Dual lattice of the lattice $\mathcal{L}$.
B	A matrix whose columns form a basis of the lattice $\mathcal{L}'$.
t	Parameter used to transform vectors returned by quantum circuit into vectors that approximates dual lattice vectors in $\mathcal{L}^*$.

References

1. R. L. Rivest, A. Shamir and L. Adleman (1978). "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2.
2. P. W. Shor (1994). "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*.
3. P. W. Shor (1999). "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, 1999.
4. D. Dziechciarz and M. Niemiec (2025). "Efficiency analysis of NIST-standardized post-quantum cryptographic algorithms for digital signatures in various environments," *Electronics*, vol. 14, no. 1.
5. R. Dhaulakhandi, B. K. Behera and F. J. Seo (2024). "Factorization of large tetra and penta prime numbers on IBM quantum processor," *APL Quantum*, vol. 1, p. 026113, 06.
6. P. M. Mutter and G. Burkard (2023). "Theory of qubit noise characterization using the long-time cavity transmission," *Physical Review A*, vol. 107, no. 2.
7. O. Regev (2025). "An efficient quantum factoring algorithm," *J. ACM*, vol. 72.
8. M. Kiebert (2024). "Oded Regev's quantum factoring algorithm." Mathematics and Computer Science at Informatics Institute, Korteweg-de Vries Institute for Mathematics, Faculty of Sciences, University of Amsterdam.
9. S. Ragavan and V. Vaikuntanathan (2024). "Space-efficient and noise-robust quantum factoring," in *Advances in Cryptology – CRYPTO 2024* (L. Reyzin and D. Stebila, eds.), (Cham), pp. 107–140, Springer Nature Switzerland.
10. M. Ekerå and J. Gärtner (2025). "A high-level comparison of state-of-the-art quantum algorithms for breaking asymmetric cryptography," *IACR Communications in Cryptology*, vol. 2, no. 1.

11. B. Stepień (2021). “Algorytm Shora dla IBM Qiskit,” Master’s thesis, AGH University of Krakow, Krakow, Poland. [in Polish].
12. S. Beauregard (2003). “Circuit for Shor’s algorithm using $2n + 3$ qubits,” *arXiv preprint quant-ph/0205095*.
13. Y. Takahashi (2008). *Efficient quantum circuits for arithmetic operations and their applications*. PhD thesis, The University of Electro-Communications, Tokyo.
14. T. Häner, M. Roetteler and K. M. Svore (2017). “Factoring using $2n + 2$ qubits with Toffoli based modular multiplication.” URL: <https://arxiv.org/abs/1611.07995>.
15. Y. Takahashi and N. Kunihiro (2006). “A quantum circuit for Shor’s factoring algorithm using $2n + 2$ qubits,” *Quantum Information & Computation*, vol. 6, no. 2.
16. X. Tan and P. Gao (2024). “An efficient quantum circuit implementation of Shor’s algorithm for GPU accelerated simulation,” *AIP Advances*, vol. 14, no. 2.
17. R. Cleve and J. Watrous (2000). “Fast parallel circuits for the quantum Fourier transform,” in *Proc. 41st Annual Symposium on Foundations of Computer Science*.
18. D. Harvey and J. Van Der Hoeven (2021). “Integer multiplication in time $O(n \log n)$,” *Annals of Mathematics*, vol. 193, no. 2.
19. C. Gidney (2019). “Asymptotically efficient quantum Karatsuba multiplication,” *arXiv preprint arXiv:1904.07356*.
20. G. D. Kahanamoku-Meyer and N. Y. Yao (2024). “Fast quantum integer multiplication with zero ancillas,” *arXiv preprint arXiv:2403.18006*.
21. C. Zalka (2006). “Shor’s algorithm with fewer (pure) qubits,” *arXiv preprint quant-ph/0601097*.
22. C. Gidney (2017). “Factoring with $n + 2$ clean qubits and $n - 1$ dirty qubits,” *arXiv preprint arXiv:1706.07884*.
23. M. Roetteler, M. Naehrig, K. M. Svore and K. Lauter (2017). “Quantum resource estimates for computing elliptic curve discrete logarithms,” in *Advances in Cryptology—ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security*.
24. A. Lenstra, H. Lenstra and L. László (1982). “Factoring polynomials with rational coefficients,” *Mathematische Annalen*, vol. 261.
25. C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant (2020). “Array programming with NumPy,” *Nature*, vol. 585.
26. A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson and J. M. Gambetta (2024). “Quantum computing with Qiskit.” *arXiv 10.48550/arXiv.2405.08810*.
27. Z. Cai, R. Babbush, S. C. Benjamin, S. Endo, W. J. Huggins, Y. Li, J. R. McClean and T. E. O’Brien (2023). “Quantum error mitigation,” *Reviews of Modern Physics*, vol. 95, no. 4.
28. IBM (2024). “Error mitigation and suppression techniques.” URL: <https://quantum.cloud.ibm.com/docs/en/guides/error-mitigation-and-suppression-techniques>, Accessed on March 17, 2026.
29. O. Regev (2024). “An efficient quantum factoring algorithm quantum colloquium.” URL: <https://simons.berkeley.edu/events/efficient-quantum-factoring-algorithm-quantum-colloquium>.