International Journal of Theoretical Physics (2024) 63:14
https://doi.org/10.1007/s10773-023-05532-4

RESEARCH

On the Various Ways of Quantum Implementation of the Modular Exponentiation Function for Shor's Factorization

Jiří Tomčala[1]

Received: 27 January 2023 / Accepted: 18 December 2023 / Published online: 10 January 2024

Abstract
The content of this paper is a detailed analysis of possible ways how to quantum implement a key part of Shor's factorization algorithm, the modular exponentiation function. This implementation is a bottleneck for performing quantum factorization with polynomial complexity, which would make it possible to factorize really large numbers in a reasonable amount of time. In this paper, not only the general theory is presented, but also the results of successful factorizations of the numbers 247 and 143 using Shor's algorithm from a quantum computer simulator. An interesting fact is that no ancillary qubits were needed in these factorizations. Based on the content of the paper, the conclusion also suggests possible future work on the development of this modular exponentiation function implementation.

Keywords Quantum factorization · Shor's algorithm · Modular exponentiation function · Breaking RSA · Quantum implementation

1 Introduction

A very widespread system for securing electronic communications, the RSA (Rivest-Shamir-Adleman) cryptosystem [1], [2] is based on the fact that factorizing a sufficiently large number (the public key) requires a huge amount of time. Currently, a public key of 2048 bits is commonly used, which would theoretically take tens of millions of years to factorize even on today's supercomputers.

This is because even the fastest classical algorithms perform factorization in sub-exponential time. However, using Shor's algorithm with a quantum circuit, this factorization can theoretically be done in polynomial time.

This algorithm was developed in 1994 by the American mathematician Peter Shor. In 2001, this algorithm was practically verified [3] by a research group at IBM that factored the number 15 into factors 3 and 5 using an NMR implementation of a 7-qubit quantum com-

✉ Jiří Tomčala
jiri.tomcala@vsb.cz

[1] IT4Innovations, VSB – Technical University of Ostrava, 17.listopadu 2172/15, Ostrava-Poruba 70833, Czech Republic

puter. Following this implementation, two independent groups [4] [5] implemented Shor's algorithm using photonic qubits, finding that multiple qubits were observed to be entangled when running a Shor's algorithm circuit. In 2012, a successful factorization of the number 21 was performed [6] using this algorithm and in 2019 a failed attempt was made [7] to factor the number 35 on the IBM Q System One.

The principle of Shor's algorithm [8] has been described many times in the scientific literature [9] [10] [11] and therefore will be recalled here very briefly. This algorithm can also be used without a quantum circuit, but only with it can the factorization be done in polynomial time, as mentioned. The task of the quantum circuit is to find the period of the modular exponentiation function $f(x) = a^x \mod N$. This can involve many calculations of the function f when using a classical computer, but when using a quantum computer it is enough to measure the histogram at the output of a quantum circuit.

Simply described, the procedure of Shor's algorithm is as follows. Let the factorized number be denoted by N and let some integer a be chosen such that $1 < a < N$. Then it is necessary to somehow find the period r of the modular exponentiation function $f(x) = a^x \mod N$. If the period r is odd, it is necessary to go back and choose a different value for the a variable. Once a value of a is found for which the period r of the function f is even, it remains to test whether the inequality $a^{r/2} \neq -1 \mod N$ holds. If not, it is again necessary to go back and choose a different value for the integer a. Once the values of a and r that satisfy the above conditions are found, then the factors p and q of N can be calculated as $\gcd(a^{r/2} \pm 1, N)$.

The general shape of the quantum circuit for determining the period of the function f is based on the quantum phase estimation circuit [12] and is shown in Fig. 1.

The implementation of the inverse quantum Fourier transform ($QFT^\dagger$), Hadamard gates (H) and gate X is quite seamless. The depth of the QFT (and also $QFT^\dagger$) circuit grows linearly with the number of qubits and even only logarithmically in special implementations [13]. However, the biggest challenge is the implementation of the oracle U_f, which is what this article deals with. The required dependence between the input and output of the oracle U_f could be described by the following relation

$$U_f(|x\rangle \otimes |0 \ldots 01\rangle) = |x\rangle \otimes |a^x \mod N\rangle. \tag{1}$$

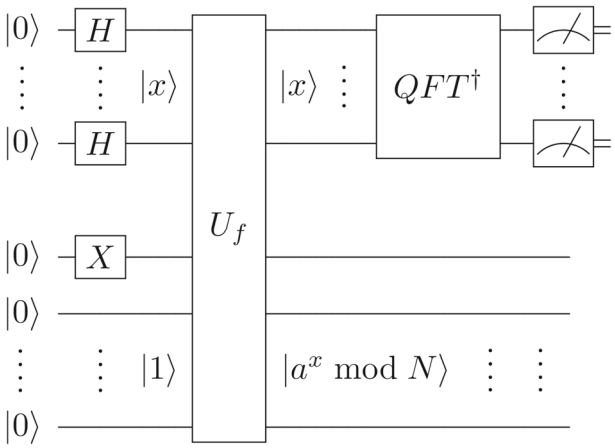

Fig. 1 General period-finder circuit for Shor's algorithm

Thus, it is clear from the above that the key component of the oracle U_f, and thus of the entire circuit in Fig. 1, is the implementation of the modular exponentiation function $a^x \bmod N$. In the following chapters, various ways of this implementation are discussed.

2 Implementation Using Basic Arithmetic Operations

Probably the very first way one can think of is to implement it using a composition of implementations of basic arithmetic operations. This way of implementation is detailed in [14]. To give an idea of how complex such a quantum circuit would be, Fig. 2 shows a simple example of a quantum implementation of a 3-qubit adder.

As can be seen from Fig. 2, to implement such a simple mathematical operation, a quantum circuit of theoretical depth 10 is required, plus 8 ancillary qubits. If n denotes the number of bits needed to binary represent a factorized number N, then a quantum circuit with a depth of $400n^3 - 400n^2 + 75n$ and a total of $7n + 1$ qubits is needed to implement the entire modular exponentiation function [14].

Today's quantum computers already have tens of qubits, so they would meet this requirement, but running a quantum circuit with that much depth would, due to decoherence of their qubits [15], lead to measuring completely unusable results. For example, to factorize a relatively small number $N = 247$ ($n = 8$), the depth of the quantum circuit constructed in

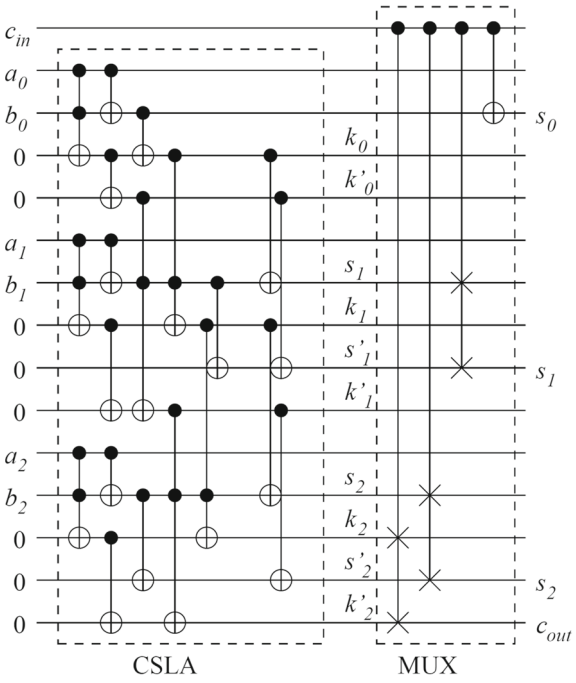

Fig. 2 3-qubit carry-select adder(CSLA) with multiplexer(MUX). The result is $s = a + b$. Figure adapted from [14]

this way would be 179800 and its width 57 qubits. This is way beyond the reach of current quantum computers.

3 Implementation Utilizing Rotations in the Fourier Domain

Arithmetic operations can also be performed in the Fourier domain by rotating the quantum states of individual qubits. Since the quantum Fourier transform as well as quantum state rotation are relatively cheap operations [13] for a quantum computer, this approach makes some sense.

The quantum Fourier transform [16] converts the input binary combination of quantum states $|0\rangle$ and $|1\rangle$ into the output combination of phase shifts of superposed quantum states $|+\rangle$. Various operations can then be performed by additional rotations, see Fig. 3. Once all operations are complete, the combination of phase shifts can then be converted back to a binary combination using the inverse quantum Fourier transform, which is also simple to implement, as is the direct one mentioned above.

As can be seen from the general implementation of the adder in Fig. 3, a quantum circuit to add two 3-qubit numbers (as in Fig. 2) would have a theoretical depth of only 3 and does not need any extra ancillary qubits. However, to the depth of this circuit it is necessary to add the depth of the direct and inverse quantum Fourier transform. But these are only required at the beginning and at the end of the whole calculation.

To implement the entire modular exponentiation function in this way, a $(9n + 2)$-qubit quantum circuit with a depth of only $2000n^2$ is then required [17]. A 74-qubit quantum circuit with a depth of 128000 would then need to be run to factorize number $N = 247$ ($n = 8$).

Compared to the previous method, this is a slightly simpler circuit, but still not applicable to today's quantum computers.

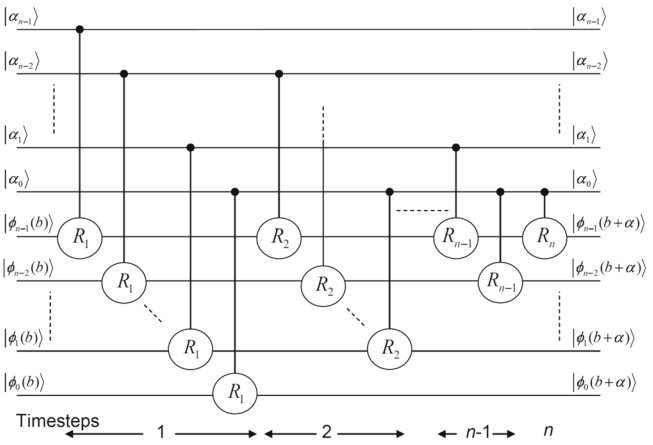

Fig. 3 General implementation of the adder utilizing rotations in the Fourier domain. The result is $b + \alpha$. The scheme does not contain a direct nor an inverse quantum Fourier transform. R_k are controlled Z-rotations by angle $2\pi/2^k$. Figure adapted from [17]

4 Implementation by Creating a Specially Designed Circuit

Another option is not to create a quantum circuit implementing a general modular exponentiation function, but always to create a special circuit for this function with specific values of the parameters a and N.

Such a circuit does not necessarily have to implement all possible input/output combinations, but only a few necessary ones that follow each other and start at a certain initial input value, in this case the number 1. This simplification occurred to me while studying an interesting paper [18] from 2012 on a similar topic. Interestingly, another paper [19], which was published during the long review process of this paper, also uses the same principle to implement the modular exponentiation function. So this is not some revolutionary idea, but rather just an intuitive approach to simplify this implementation.

Since the function $f(x) = a^x \mod N$ can be decomposed into a product

$$a^x \mod N = (...((a^{x_0 2^0} \mod N) \times a^{x_1 2^1} \mod N) \times ...) \times a^{x_{n-1} 2^{n-1}} \mod N, \quad (2)$$

where $(x_{n-1}x_{n-2} \ldots x_2 x_1 x_0)_2$ is the binary representation of the variable x, which means that

$$x = x_0 2^0 + x_1 2^1 + x_2 2^2 + ... + x_{n-1} 2^{n-1} \ \ (x_0, x_1, x_2, ..., x_{n-1} \in \{0, 1\}), \quad (3)$$

then the entire function $f(x) = a^x \mod N$ can be implemented as shown in Fig. 4. This figure shows an example circuit for specific values of a and N. The conditions for triggering each multiplication are the upper qubits that represent the variable x.

The number of these upper qubits determines the accuracy with which the period is found. A higher number of these qubits would result in a more accurate detection of the

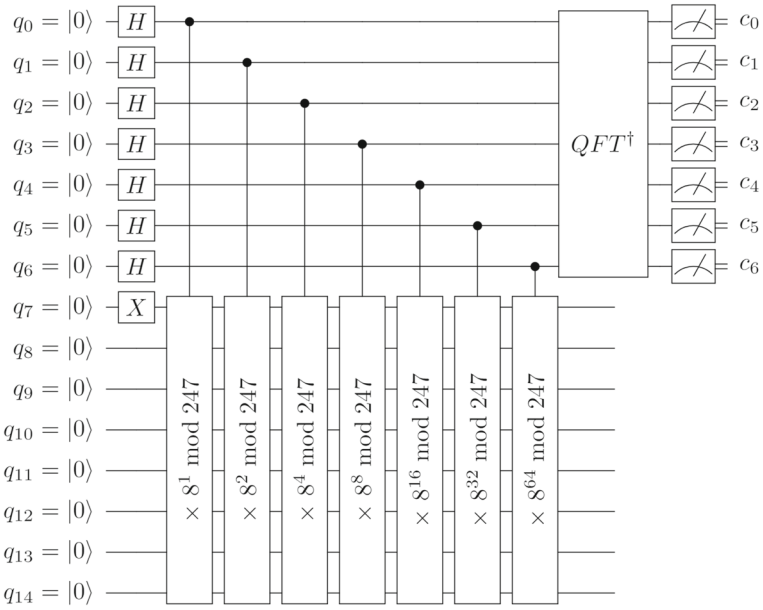

Fig. 4 Specific period-finder circuit for $f(x) = 8^x \mod 247$. The upper 7 qubits $q_0 - q_6$ are used to represent the variable x and then to measure the period of the function f. The lower 8 qubits $q_7 - q_{14}$ are needed to represent the results of consecutive multiplications, which by definition never exceed $N - 1 = 246$, so 8 qubits are sufficient

period searched, but on the other hand it would also increase the depth of the whole circuit. It is therefore appropriate to choose the optimum number of them. The numerical results in Section 5 show that 7 qubits ($q_0 - q_6$) are enough to achieve satisfactory accuracy.

Based on the above, it is sufficient to implement the function $g(y) = (y \times a) \bmod N$ and then implement the composite functions

$$\begin{aligned}
&(y \times a^2) \bmod N = (((y \times a) \bmod N) \times a) \bmod N = g(g(y)),\\
&(y \times a^3) \bmod N = ((((y \times a) \bmod N) \times a) \bmod N) \times a) \bmod N = g(g(g(y))),\\
&\quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots
\end{aligned} \tag{4}$$

etc. by simply repeating the circuit implementing the function g in series, as illustrated by the specific examples in Fig. 5.

Now all that remains is to create a quantum circuit implementing the g function itself. For the purpose of finding the period of the function f, it is sufficient if this implementation of the function g works only for one particular input number and also for all other, subsequent numbers that may arise during its successive multiplication. An illustrative, specific example of these input/output numbers is shown in Table 1.

The number 1 was chosen as the initial input number in this table, which corresponds to the circuit in Fig. 4, where this initial input value is set by X-gate on qubit q_7. Any initial input number (except 0) can be chosen here, but for the sake of simpler implementation of the g function, the number 1 was chosen.

Based on Table 1, the corresponding quantum circuit can then be constructed. One possible composition of this circuit, which was also used to measure the results presented in Section 5, is shown in Fig. 6.

The big advantage of this implementation method over the previous two is the much smaller depth of the resulting circuit and the fact that there is no need for any ancillary qubits. The depth of the quantum circuit in Fig. 6, implementing the g function, is only 21. The implementation of the entire modular exponentiation function f for this particular example then leads to a quantum circuit of depth $21 \times (1+2+4+8+16+32+64) = 2667$ with a total number of qubits n, which in this case is just 8.

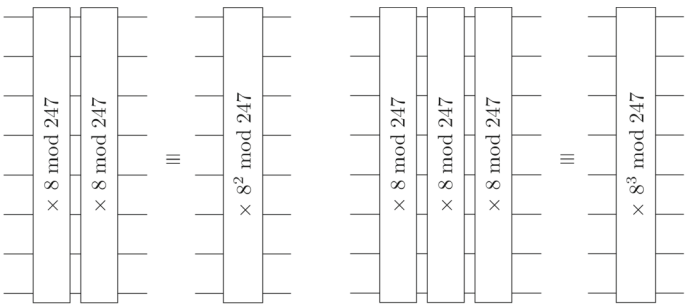

Fig. 5 An example of how to construct the specific composite functions $(y \times 8^2) \bmod 247 = g(g(y))$, $(y \times 8^3) \bmod 247 = g(g(g(y)))$, etc. from function $g(y) = (y \times 8) \bmod 247$

Table 1 Function $g(y) = (y \times 8) \bmod 247$ truth table for an initial input value of 1

Input		Output	Input		Output
0000 0001 (1)	→	0000 1000 (8)	0100 1101 (77)	→	0111 1010 (122)
0000 1000 (8)	→	0100 0000 (64)	0111 1010 (122)	→	1110 1011 (235)
0100 0000 (64)	→	0001 0010 (18)	1110 1011 (235)	→	1001 0111 (151)
0001 0010 (18)	→	1001 0000 (144)	1001 0111 (151)	→	1101 1100 (220)
1001 0000 (144)	→	1010 0100 (164)	1101 1100 (220)	→	0001 1111 (31)
1010 0100 (164)	→	0100 1101 (77)	0001 1111 (31)	→	0000 0001 (1)

But this is only a theoretical depth, because in the real implementation in the Qiskit framework [20], there are no 8-qubit, 7-qubit, or 6-qubit gates available. So these gates were decomposed into several less-qubit gates, see Fig. 7.

The depth of the circuit in Qiskit can be determined using the *depth()* function, which found a depth of 34 for this circuit. The total depth of the modular exponentiation function implemented in this way in Qiskit is then $34 \times (1 + 2 + 4 + 8 + 16 + 32 + 64) = 4318$.

For factorization of some other numbers, the implementation of modular exponentiation function by this method can be even simpler. An example of such a case is number 143. If the value 21 is chosen as the variable a, then the truth table of the function $g(y) = (y \times 21) \bmod 143$ comes out very simply, see Table 2.

This is of course reflected in the relative simplicity of the quantum circuit implementing this function, which is shown in Fig. 8. Only 7 qubits are needed and as can be seen from the picture, the theoretical depth of this circuit is just 8. The implementation of the entire modular exponentiation function $f(x) = 21^x \bmod 143$ then has a theoretical depth of $8 \times (1 + 2 + 4 + 8 + 16 + 32 + 64) = 1016$.

When implemented in Qiskit, some gates were again broken down into several smaller gates and the result is shown in Fig. 9. The depth was again determined using the *depth()* function, which found a depth of 15 for this circuit. The total depth of the modular exponentiation function implemented in this way in Qiskit then appears to be $15 \times (1 + 2 + 4 + 8 + 16 + 32 + 64) = 1905$, however, an interesting situation has arisen here, because when serially repeating this circuit, the last gate (X) can be performed simultaneously with the first gate (CNOT) of another copy of the same circuit. So the total real depth is then $1905 - 0 - 1 - 3 - 7 - 15 - 31 - 63 = 1785$.

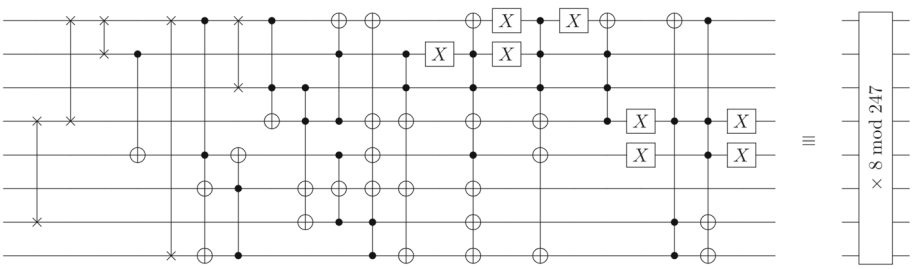

Fig. 6 A quantum circuit assembled according to the required inputs and outputs listed in Table 1

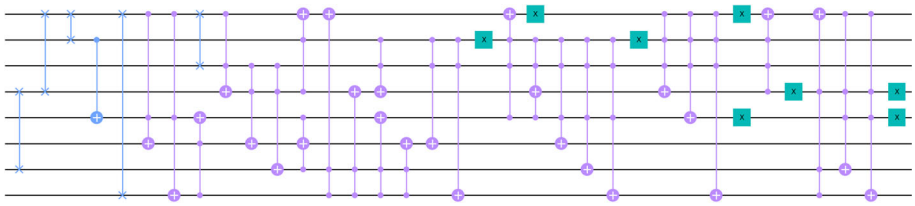

Fig. 7 Quantum circuit from Fig. 6 implemented in Qiskit framework

Table 2 Function $g(y) = (y \times 21) \bmod 143$ truth table for an initial input value of 1

Input	Output
000 0001 (1)	001 0101 (21)
001 0101 (21)	000 1100 (12)
000 1100 (12)	110 1101 (109)
110 1101 (109)	000 0001 (1)

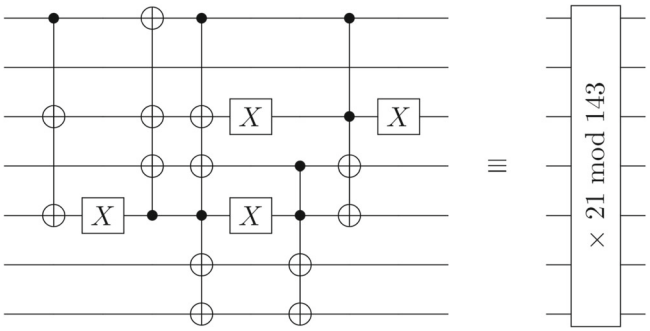

Fig. 8 A quantum circuit assembled according to the required inputs and outputs listed in Table 2

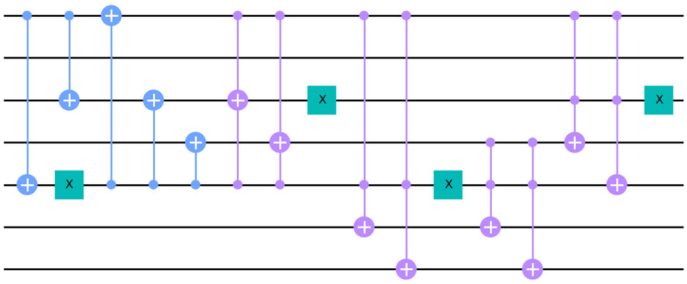

Fig. 9 Quantum circuit from Fig. 8 implemented in Qiskit framework

As for the size of the $\mathrm{QFT}^{\dagger}$ circuit, in both cases of factorization its width was 7 qubits and its depth was 14.

Constructing all these circuits according to the required inputs and outputs from the truth table can be done in several ways. This vast topic would be worth a separate paper, however it is at least worth mentioning that recently an interesting idea has emerged to build these circuits using evolutionary algorithms, see [21], [22].

In general, every modular exponentiation function of the form $f(x) = a^x \bmod N$ is inherently periodic and bounded. This is due to the modulo operation, which ensures that $f \in \langle 1, N-1 \rangle$ always holds. Thus, the number of values of the function f is finite and sooner or later it will happen that $f(x) = f(x+r)$. Then with simple adjustments it can be shown that $f(x+1) = f(x+r+1)$ also holds:

$$\begin{aligned} f(x+1) &= a^{x+1} \bmod N = (a^x \times a) \bmod N = ((a^x \bmod N) \times a) \bmod N \\ &= (f(x) \times a) \bmod N = (f(x+r) \times a) \bmod N = ((a^{x+r} \bmod N) \times a) \bmod N \\ &= (a^{x+r} \times a) \bmod N = a^{x+r+1} \bmod N = f(x+r+1). \end{aligned} \tag{5}$$

This proves that the function f is always periodic. Therefore, for each possible combination of a and N, a circuit that implements such a function $f(x) = a^x \bmod N$ can always be constructed. Since each such specific circuit covers a much smaller number of possible input and output states than some universal circuit, a reduction in the number of gates, circuit depth, and the number of qubits required compared to any universal circuit is always guaranteed.

Compared to the previous two methods that try to create such a universal circuit, both resulting modular exponentiation function implementations (Figs. 7 and 9) have about two orders of magnitude less depth, so it could be run on a quantum computer simulator and the results are presented in the following section.

5 Results

Due to the excessive depth of the circuits resulting from the first two methods, it was only possible to run the entire quantum circuit constructed by the third method, i.e. where the modular exponentiation function is implemented by creating a specially designed circuit. First, it should be mentioned that all histograms of simulated quantum circuits were obtained using 100,000 shots. The assembled quantum circuit from Fig. 4 was run on IBM quantum computer simulator *ibmq_qasm_simulator* and the resulting histogram is shown in Fig. 10. Initial simulations of this circuit were first run with non-zero quantum noise set, but this was such a challenging task that results could not be in IBM Quantum Lab [20] obtained in a

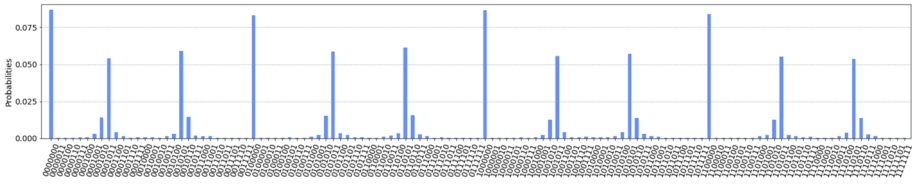

Fig. 10 Resulting histogram of period-finder circuit for $f(x) = 8^x \bmod 247$ obtained using Qiskit framework [20] and IBM Quantum simulator *ibmq_qasm_simulator* without any quantum noise. Number of shots: 100,000

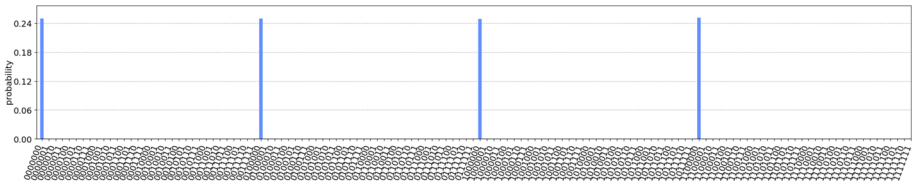

Fig. 11 Resulting histogram of period-finder circuit for $f(x) = 21^x \mod 143$ obtained using Qiskit framework [20] and IBM Quantum simulator *ibmq_qasm_simulator* without any quantum noise. Number of shots: 100,000

reasonable amount of time. So the figure presents the resulting histogram obtained in the simulation without any quantum noise.

The histogram shows the probability distribution of all possible output states that would be measured on qubits $(q_0 - q_6)$. From this histogram, the period $r = 12$ can be determined using continued fractions [12]. In this case, this period can also be easily determined by the number of peaks in this histogram. Once the period r is known, the factors can simply be determined as $\gcd(a^{r/2} \pm 1, N) = \gcd(8^6 \pm 1, 247) = \gcd(262144 \pm 1, 247)$, which are the numbers 13 and 19. By simple multiplication of these two factors found $(13 \times 19 = 247)$, it is easy to ascertain the correctness of the whole factorization.

When factoring the number 143 with a smaller circuit for the function $f(x) = 21^x \mod 143$, the simulation succeeded not only without any quantum noise, but also with the depolarization set to the probability values 0.00005 for a 1-qubit gate, 0.0001 for a 2-qubit gate, and 0.0002 for a 3-qubit gate. The results are shown in Figs. 11 and 12. From the measured histograms, the period $r = 4$ can be determined. Individual factors can then be again easily calculated from this value r: $\gcd(a^{r/2} \pm 1, N) = \gcd(21^2 \pm 1, 143) = \gcd(441 \pm 1, 143)$, which are the numbers 11, 13, and since their product is 143, the factors found are correct.

6 Conclusion & Future Work

The main conclusion is that the only way to currently use Shor's factorization algorithm is to implement the modular exponentiation function with a specially designed circuit for each particular case. It was also shown that for each specific factorization such a special circuit can always be created and thus this factorization can be always performed at least on a simulator. Unfortunately, the circuits resulting from the first (using basic arithmetic operations) and second (utilizing rotations in the Fourier domain) method are potentially so large that they are at the limit of being runnable on a simulator, let alone a real quantum computer. Their

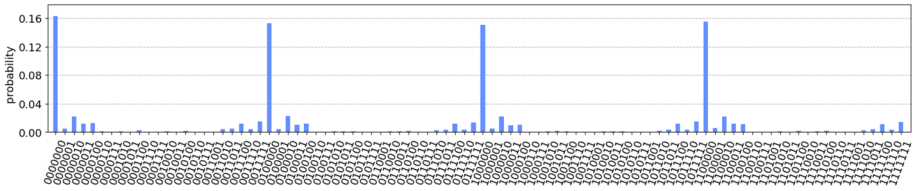

Fig. 12 Resulting histogram of period-finder circuit for $f(x) = 21^x \mod 143$ obtained using Qiskit framework [20] and IBM Quantum simulator *ibmq_qasm_simulator* with depolarization quantum noise set with error probability parameters of 0.00005 for a 1-qubit gate, 0.0001 for a 2-qubit gate, and 0.0002 for a 3-qubit gate. Number of shots: 100,000

depth ranges in the order of hundreds of thousands, while their required width exceeds 50 qubits.

The important result is that using the third method (with a specially designed circuit), the quantum period-finder circuits were successfully run on the simulator and thereby factorized the numbers 247 and 143. The key advantage is that these factorizations did not need any ancillary qubits. This allows for more efficient use of an eventual real quantum device.

So far, such high numbers have not yet been factorized using Shor's algorithm on a quantum computer. Larger numbers have been factored using variational quantum algorithms, but they proceed iteratively with the help of an optimizer, running on a classical computer. Thus, compared to a single run of the period-finder circuit in Shor's factorization, variational factorization requires multiple runs of the variational quantum circuit during the iterations and, in addition, optimization of the parameters on a classic computer must take place between them.

The direction of further work would be experimental simulations with different settings of quantum noise, so that it would be possible to find out at which parameters of quantum noise the results of factorization are still sufficiently accurate. The challenge for future work is also to find some general way of creating specially designed quantum circuits implementing the modular exponentiation function, and also to reduce the depth of these circuits to the point where they give usable results when run on a real quantum computer. Today's quantum computers are capable of running quantum circuits with depths of at most the order of tens. Given the rapid development of quantum technologies, which make it possible to run larger and larger quantum circuits, quantum factorization of even larger numbers should be possible in the future.

Acknowledgements This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254).

Author Contributions J.T. wrote all the text and prepared all the pictures.

Funding Open access publishing supported by the National Technical Library in Prague.

Declarations

Competing interests The authors declare no competing interests.

References

1. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978). https://doi.org/10.1145/359340.359342
2. Nisha, S., Farik, M.: Rsa public key cryptography algorithm - a review. Int. J. Sci. Technol. Res. **6**, 187–191 (2017)

3. Vandersypen, L.M.K., Steffen, M., Breyta, G., Yannoni, C.S., Sherwood, M.H., Chuang, I.L.: Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. Nature **414**, 883–887 (2001). https://doi.org/10.1038/414883a
4. Lu, C.-Y., Browne, D.E., Yang, T., Pan, J.-W.: Demonstration of a compiled version of Shor's quantum factoring algorithm using photonic qubits. Phys. Rev. Lett. **99**, 250504 (2007). https://doi.org/10.1103/PhysRevLett.99.250504
5. Lanyon, B.P., Weinhold, T.J., Langford, N.K., Barbieri, M., James, D.F.V., Gilchrist, A., White, A.G.: Experimental demonstration of a compiled version of Shor's algorithm with quantum entanglement. Phys. Rev. Lett. **99**, 250505 (2007). https://doi.org/10.1103/PhysRevLett.99.250505
6. Martín-López, E., Laing, A., Lawson, T., Alvarez, R., Zhou, X., O'Brien, J.L.: Experimental realization of Shor's quantum factoring algorithm using qubit recycling. Nature Photonics **6**, 773–776 (2012). https://doi.org/10.1038/nphoton.2012.259
7. Amico, M., Saleem, Z.H., Kumph, M.: Experimental study of Shor's factoring algorithm using the IBM Q Experience. Phys. Rev. A **100**, 012305 (2019). https://doi.org/10.1103/PhysRevA.100.012305
8. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science (1994)
9. Ugwuishiwu, C., Orji, U., Ugwu, C., Asogwa, C.: An overview of quantum cryptography and Shor's algorithm. Int. J. Adv. Trends Comput. Sci. Eng **9**(5) (2020)
10. Yimsiriwattana, A., Lomonaco Jr, S.J.: Distributed quantum computing: A distributed Shor algorithm. In: Quantum information and computation II, vol. 5436, pp. 360–372 (2004). SPIE
11. Bhatia, V., Ramkumar, K.: An efficient quantum computing technique for cracking RSA using Shor's algorithm. In: 2020 IEEE 5th international conference on computing communication and automation (ICCCA), pp. 89–94 (2020). IEEE
12. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th, Anniversary Cambridge University Press, Cambridge (2010)
13. Cleve, R., Watrous, J.: Fast parallel circuits for the quantum fourier transform. In: Proceedings 41st annual symposium on foundations of computer science, pp. 526–536 (2000). https://doi.org/10.1109/SFCS.2000.892140
14. Van Meter, R., Itoh, K.M.: Fast quantum modular exponentiation. Phys. Rev. A **71**, 052320 (2005). https://doi.org/10.1103/PhysRevA.71.052320
15. Schlosshauer, M.: Quantum decoherence. Physics Reports **831**, 1–57 (2019)
16. Moore, C., Rockmore, D., Russell, A.: Generic quantum Fourier transforms. ACM Transactions on Algorithms (TALG) **2**(4), 707–723 (2006)
17. Pavlidis, A., Gizopoulos, D.: Fast quantum modular exponentiation architecture for Shor's factorization algorithm.Quantum information & computation 14,649–682 (2014)
18. Markov, I., Saeedi, M.: Constant-optimized quantum circuits for modular multiplication and exponentiation. Quantum Information and Computation **12**(2012). https://doi.org/10.26421/QIC12.5-6-1
19. Singleton Jr, R.L.: Shor's factoring algorithm and modular exponentiation operators. Quanta**12**(1),41–130(2023). https://doi.org/10.12743/quanta.v12i1.235
20. IBM Quantum various authors: Qiskit: An Open-source Framework for Quantum Computing (2022)
21. Creevey, F.M., Hill, C.D., Hollenberg, L.C.L.: GASP: a genetic algorithm for state preparation on quantum computers. Scientific Reports **13**, 11956 (2023). https://doi.org/10.1038/s41598-023-37767-w
22. Zelinka, I., Kojecký, L., Lampart, M., Nowaková, J.,Plucar, J.: iSOMA swarm intelligence algorithm in synthesis of quantum computing circuits.Applied Soft Computing 142,110350 (2023). https://doi.org/10.1016/j.asoc.2023.110350