D. Demmler, D. Krupka, H. Federrath. (Hrsg.): INFORMATIK 2022,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2022 1133

Implementations for Shor's algorithm for the DLP

Alexander Mandl[1], Uwe Egly[2]

Abstract: Shor's algorithm for solving the discrete logarithm problem is one of the most celebrated works in quantum computing. It builds upon a quantum circuit performing modular exponentiation. As this is a comparatively expensive process, many approaches for reducing both the number of used qubits and the number of applied gate operations have been proposed. We provide quantum circuits in Qiskit for three different implementation proposals aiming to reduce space complexity and compare their performance regarding their asymptotic gate complexity. We make use of the circuit implementations and Qiskit's simulation capabilities to compare the actual number of applied gate operations in compiled circuits for small problem instances to aid future applications of this algorithm.

Keywords: quantum computing; discrete logarithm; implementation; Shor's algorithm

1 Introduction

Shor's polynomial-time algorithms for prime factorization and the discrete logarithm problem (DLP) [Sh97] are among the most groundbreaking results in the field of quantum computation. These problems are of particular interest since their hardness is the basis of the security of various cryptographic systems, because the problems are believed to not be solvable on a classical computer in polynomial time. Many variants and reformulations [ME98, PZ03, Ka17] of Shor's algorithms have since been studied and, although the physical realization of quantum computers powerful enough to execute them for meaningful inputs still lies in the future, the field of quantum computation has grown considerably.

Shor's algorithms make use of the fact that the quantum Fourier transform (QFT) can be used to estimate the eigenvalues of quantum transformations in a process referred to as quantum phase estimation (QPE) [ME98]. This operation uses a quantum oracle which computes some unitary operation U which is known to have the eigenvector $|\psi\rangle$ and the corresponding eigenvalue $e^{2\pi i\phi}$. The output of the overall process is then the closest integer to $\phi 2^n$ for n depending on the quantum circuit size. Furthermore, it can be shown that this value is obtained with a probability of at least $4/\pi^2$ [ME98].

We focus on the DLP which aims to find an integer m such that $g^m \equiv b \bmod p$ for a given integer-valued generator g, an integer b, and a prime number p. The quantum oracle

[1] Institute of Architecture of Application Systems, University of Stuttgart, Germany, alexander.mandl@iaas.uni-stuttgart.de
[2] Institute of Logic and Computation, TU Wien, Austria, uwe.egly@tuwien.ac.at

doi:10.18420/inf2022_96

involved in the QPE process is a quantum circuit U_c that performs the modular multiplication $U_c |y\rangle = |cy \bmod p\rangle$.

Since the QPE algorithm requires the repeated controlled application of this oracle for different constants, which amounts to the exponentiation $|x\rangle U_{c^x} |y\rangle = |x\rangle |c^x y \bmod p\rangle$, it is usually referred to as the *modular exponentiation* oracle. In the description of this circuit it is assumed that the constant c is built into the circuit.

Given the multiplicative order of the generator g modulo p, described by $r = |\langle g \rangle|$, the eigenvalues of this transformation have the form

$$U_b |\psi_k\rangle = e^{\frac{2\pi i k m}{r}} |\psi_k\rangle ,$$

for $0 \leq k < r$, with eigenvectors $|\psi_k\rangle$. Herein, b refers to the input of the DLP from above.

The algorithm uses QPE to estimate the eigenvectors for both the oracle U_b performing the multiplication with b modulo p and U_g which performs the same operation with the integer g and has eigenvalues $\exp(2\pi i k/r)$. This leads to estimates of the fractions $(km \bmod r)/r$ as well as k/r, which can be used to infer the discrete logarithm m on a classical computer.

The fundamental result of Shor's work is that the best approximations to these two fractions are obtained with a constant probability greater than zero, which implies that the quantum circuit provides the correct measurements after a constant number of repetitions. For the purpose of inferring m it has to hold that the measured k and the order r have to be coprime for the algorithm to succeed. However, for large r it was shown that a lower bound for this probability can be given [Sh97], which results in a total runtime for the whole algorithm of

$$O((\log \log p) T(n)), \tag{1}$$

where $T(n)$ describes the number of gate operations required to perform the oracle operation.

Although Shor's algorithm succeeds in solving the DLP in a theoretical setting, the application of this process using real machines still remains an open problem. Currently, there are no quantum computers powerful enough to run this algorithm even for relatively small problem instances of a few bits. This stems from the fact that the involved transformations have to use a number of extra temporary qubits, so-called *ancilla* qubits, which greatly increase the resource demand of the whole algorithm beyond the capabilities of most current quantum computers. On the other hand, the sheer number of operations both in the modular exponentiation oracle as well as in the overall algorithm introduces the possibility for errors in the physical realization of the quantum transformations, which leads to unusable measurement results even for moderately small instances. Both of these obstacles are further highlighted by the algorithm descriptions in the later parts of this text.

However, the steady increase in processing power of quantum computers might lead to future machines powerful enough to run the algorithm. To aid future realizations of Shor's algorithm, this work provides implementations[3] of the algorithm for the discrete logarithm

[3] The source code can be found at `https://github.com/mhinkie/ShorDiscreteLog`

problem in Qiskit[4] that focus on conserving resources both in the number of qubits and gate operations that are applied.

As described above, the modular exponentiation oracles U_{b^x} and U_{g^x} are integral parts of this quantum algorithm. Therefore, we examine efficient implementation proposals for these arithmetic operations. Implementation variants of modular multiplication on a quantum computer range from repeated addition [Be03, HRS17] to more involved approaches such as Montgomery modular multiplication or Barrett reduction [RC18]. We implement and compare three different proposals for performing this arithmetic process and present a comparison regarding both, their asymptotic gate complexity as well as regarding the actual number of applied gate operations for small examples. Furthermore, there are multiple variants of the overall phase estimation approach that save qubits by restructuring the circuit [Be03, ME98], some of which were also implemented in Qiskit for this work (see [Ma21] for details). We selected this particular set of algorithms as they focus on minimizing the space complexity of the modular exponentiation operation. This allows us to simulate the circuits for larger inputs on classical devices and will also allow users to execute the overall algorithm in the future on smaller quantum computers.

The comparison presented in Section 3 has to take into account that a decrease in space complexity, i. e. in the number of qubits, often goes hand-in-hand with an increase in time complexity and vice versa. For this reason, we not only compare the number of used qubits but also the gate complexity for large n and for small examples using compiled circuits. Of course, the actual computing time and space complexity are also influenced by a number of other factors that depend on the hardware that will execute the algorithm in the future such as the error correction routines. Due to their dependence on the actual physical implementation these factors were not studied explicitly in this work. Gidney and Ekerå [GE21] analyze these requirements and give an approximation of the number of qubits required for an error corrected implementation of Shor's algorithm on noisy systems.

2 Modular exponentiation

The modular exponentiation operation is the most costly component of the overall algorithm, as such implementations usually focus on keeping the number of required operations as small as possible. Shor notes in [Sh97] that the Schönhage-Strassen multiplication algorithm would theoretically be the fastest one for this problem for large input sizes. However, implementations of this algorithm as a quantum circuit require a significant amount of temporary storage space [Za98, RC18] which is why we do not consider this algorithm.

The most straightforward way of performing modular multiplication is by simply repeating a modular addition circuit using the binary decomposition of the quantum input integer x:

$$c \cdot x \bmod p = c2^0 \cdot x_1 + c2^1 \cdot x_2 + \cdots + c2^{n-1} \cdot x_n \bmod p. \tag{2}$$

[4] https://qiskit.org/

Therefore, before explaining the multiplication algorithm, different ways of performing modular addition and in turn modular subtraction are investigated.

2.1 Addition using the QFT

Draper [Dr00] proposes a quantum circuit that adds two n qubit integers by using the fact that the state after the QFT of some integer a is separable and can be described as

$$\bigotimes_{k=1}^{n} |\phi_k(a)\rangle \text{ , where } |\phi_k(a)\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + \exp\left(2\pi i \frac{a}{2^k}\right) |1\rangle \right).$$

The addition circuit itself applies Qiskit's phase gate P. This operation maps arbitrary states $\alpha |0\rangle + \beta |1\rangle$ to $\alpha |0\rangle + \exp(i\lambda)\beta |1\rangle$ for a real value λ supplied at the time of construction. Specifically, using $\lambda = \frac{2\pi b}{2^k}$ for a known integer b, it maps one Fourier-transformed qubit to

$$P\left(\frac{2\pi b}{2^k}\right) |\phi_k(a)\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + \exp\left(2\pi i \frac{a+b}{2^k}\right) |1\rangle \right) = |\phi_k(a+b)\rangle .$$

Performing this process for each qubit individually will give the Fourier-transformed result of the addition in linear time, assuming the input is already supplied in its Fourier-transformed representation. This circuit can be extended to perform modular addition and modular multiplication as presented by Beauregard [Be03] and in the later sections of this text.

2.2 Bitwise addition

Another way of performing arithmetic on a quantum computer is to perform bitwise manipulations of the input similar to the processes on a classical computer. There are various descriptions of bitwise addition gates on a quantum computer ([RC18] gives a good overview). We chose to examine the implementations suggested by Häner et al. [HRS17] further. Although this bitwise addition gate uses comparatively many ancilla qubits, it is able to perform the calculations using *borrowed* ancilla qubits. These are idle qubits that can start in any arbitrary state and will be returned to the same state after the computation. Since in the overall multiplication circuit many such idle qubits are present, the number of used qubits for the algorithm can be decreased this way.

As its core component, the circuit relies on a gate that calculates the final carry of an addition of two n bit integers a and b, where the latter is known at the time of construction of the circuit. Assuming the binary representation of the $n+1$ bit addition result $r = a + b$ is given as $r_{n+1} r_n \dots r_2 r_1$, this operation can be described as

$$CARRY(b) |a\rangle |g\rangle |0\rangle = |a\rangle |g\rangle |r_{n+1}\rangle .$$

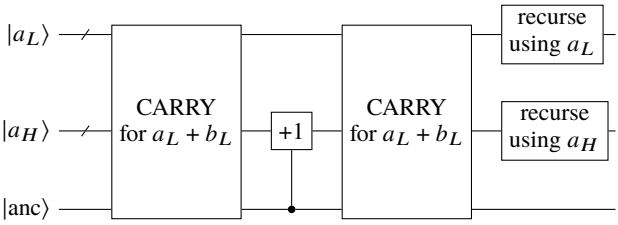

Fig. 1: Recursive addition circuit as presented in [HRS17] for a clean ancilla qubit. The adaptions required for borrowed ancilla qubits are described in [Ma21, HRS17].

The circuit needs an additional register $|g\rangle$ of size $n - 1$, which holds the borrowed qubits that are used as temporary storage during computation. How this circuit is constructed is presented in [HRS17] and details of the implementation of this process in Qiskit are presented in [Ma21]. The key concepts employed in this process will be explained here.

Using *CARRY* alone, a simple addition circuit could be constructed by iteratively computing the carry values of the addition of increasingly smaller subsets of the input with the supplied constant and toggling the resulting qubits accordingly. However, this approach would result in a circuit with $O(n^2)$ runtime [HRS17]. Häner et al. propose a different recursive approach: The n qubit input a is split into two approximately equal-sized subsets a_L and a_H with size $\lceil \frac{n}{2} \rceil$ and $\lfloor \frac{n}{2} \rfloor$ respectively. The resulting carry of the addition of a_L and the corresponding bits of the constant b_L is saved in another borrowed qubit and used to control an incrementer operation on the other half a_H of the input. The circuit then recursively computes the result for both subsets a_H and a_L to finally obtain the addition result. Figure 1 shows the corresponding quantum circuit.

The complete construction of the incrementer circuit used in each recursive step is presented in [HRS17, Ma21]. It is based on a quantum-quantum addition circuit presented in [TTK10].

The major difference between borrowed qubits as they are employed here and the zero-initialised ancilla qubits that are used in other parts of the algorithm is the fact that they can start in an arbitrary unknown state. To be able to still use these qubits to store intermediate carry values which control the application of other gates, Häner et al. reorder the operations such that they are applied before the control qubit is computed and reversed should the borrowed qubit indicate that the operation should not have been performed. To put it into other words, the operation is conditioned on the fact that this borrowed qubit is toggled.

2.3 Extension to modular addition

The modular exponentiation oracle requires a subroutine that not simply performs addition but performs addition modulo a prime module p. Using a zero-initialised ancilla qubit, both addition circuits presented here can be extended to perform modular addition by comparing the result of the addition $a + b$ with the module p.

In the bitwise adder presented in Subsection 2.2, the comparison is performed by using the *CARRY* gate to obtain the final carry of the addition $a + (b - p)$, which, as long as the register is appropriately sized, is 1 if and only if $a + b < p$. For the QFT-based adder (see Subsection 2.1), a subtraction of the module p is performed on the Fourier-transformed result before applying the inverse QFT to extract the most significant bit to decide if the subtraction by p has to be reversed.

The details of the implementation of this process in Qiskit (as well as the required uncomputation steps) can be found in [Ma21].

2.4 In-place modular multiplication

By employing the binary decomposition of the input x as presented in Equation (2), we construct a modular multiplier. This approach is described in [Be03] and similarly in [HRS17]. The circuit should compute the result $|cx \bmod p\rangle$ in the same register as the input was stored, while minimizing the number of ancillary qubits used.

The multiplication circuit is based on a circuit that performs the out-of-place multiplication $|x, y\rangle \rightarrow |x, (y + cx) \bmod p\rangle$ by repeatedly adding constants of the form $c \cdot 2^i \bmod p$ controlled by the individual input qubits $|x_i\rangle$. First the input $|x, 0\rangle$ is taken to $|x, cx \bmod p\rangle$ using the first out-of-place multiplication. After swapping both registers to obtain $|cx \bmod p\rangle\,|x\rangle$, the result will then remain in the leftmost register. The other register can be reset to the $|0\rangle$ state by performing another out-of-place multiplication with the constant $-c^{-1} \bmod p$ as the following equality shows:

$$x + (-c^{-1} \bmod p)(cx \bmod p) \bmod p = x - c^{-1}cx \bmod p = 0.$$

2.5 Montgomery modular multiplication

The quantum circuits proposed until now calculate the result of modular multiplication because the intermediate results are reduced modulo the prime p after each addition. This ensures that the number of required qubits is kept low since after each reduction the result is in the range $0 \leq x < p$. However, it also introduces unwanted overhead: The result has to be modified after each step. This overhead is especially noticeable in the case of Fourier space addition, where determining whether the result has to be reduced modulo p requires a full quantum Fourier transform and its inverse with runtime $O(n^2)$. In this section, we present the multiplication algorithm proposed by Rines and Chuang [RC18], which performs Montgomery modular multiplication [Mo85] on a quantum computer to address this problem.

As this algorithm often rearranges qubits between registers, we will highlight the size i of certain registers as $|x\rangle_i$ in the remainder of this section.

A naive optimisation of the multiplication algorithm is to use an appropriately sized register and perform all required additions on this register without ever transforming back to the computational basis. However, this implies that a division with p has to be performed at the end of the multiplication algorithm to obtain the result $|cx \bmod p\rangle$. Unfortunately, such a division requires a series of trial subtractions that are controlled by the most significant qubit of the value in the register and the extraction of this qubit again introduces a series of costly QFT operations.

Instead, the algorithm presented in [RC18] multiplies with the *Montgomery form* of the constant c, given by $c2^n \bmod p$. Afterward, the *Montgomery reduction* transforms the multiplication result $x \cdot (c2^n \bmod p)$ to $x \cdot (c2^n \bmod p)2^{-n} \bmod p$ to give the sought after modular multiplication result. The advantage of the Montgomery reduction is the fact that it does not require examinations of the most significant qubit but instead of the least significant qubit, which can be obtained in constant time by noting that

$$|\phi_1(a)\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + (-1)^{a \bmod 2}|1\rangle\right) = H|a \bmod 2\rangle\,.$$

Here, $|\phi_1(a)\rangle$ represents the least significant qubit in the Fourier transformed representation of some integer a. Therefore, a simple Hadamard gate suffices to extract this qubit in the computational basis.

In the remainder of this subsection, we give an overview of the characteristics of the quantum circuit implementing Montgomery modular multiplication. The exact specification and further information on its implementation in Qiskit can be found in [Ma21].

To distinguish the content of a whole register in the computational basis and their Fourier-transformed counterpart, the former is referred to as $|a\rangle$ and the latter is referred to as $|\phi(a)\rangle$ for some integer a.

The circuit first performs an initial multiplication of the input with the Montgomery form of the constant. This multiplication is performed by repeated addition of the Fourier-transformed register content similar to Section 2.1 and is described as

$$|x\rangle_n|0\rangle_{2n+1} \rightarrow |x\rangle_n|\phi(x \cdot (c2^n \bmod p))\rangle_{2n+1}\,.$$

2.5.1 Montgomery reduction

Given the result of the multiplication as $t = x \cdot (c2^n \bmod p)$, the *estimation stage* of the circuit first computes the values $u = tp^{-1} \bmod 2^n$ and $s = (t - up)/2^n$. It can be shown that s is then congruent to and within p of the desired result $t2^{-n} \bmod p$ [Ma21]. Therefore only one correcting addition is needed at the end of this process to obtain the result.

In each step of the quantum circuit for the estimation stage, the currently least significant qubit of the register containing $|\phi(t)\rangle$ is removed and assigned as the new most significant

qubit of the register containing $|u\rangle$, which implicitly divides $|\phi(t)\rangle$ by two. This qubit is then used to control a subtraction by $p/2$. Therefore, when $|\phi(t')\rangle$ is the content of the first register at the beginning of this iteration, the ith iteration transforms $|\phi(t')\rangle_l |u\rangle_{i-1}$ to

$$\left|\phi\left(\frac{t'}{2} - \frac{u_i \cdot p}{2}\right)\right\rangle_{l-1} |u\rangle_i ,$$

with l such that $l + (i-1) = 2n + 1$.

After n repetitions of this process, the system reaches the desired state $|\phi(s)\rangle_{n+1} |u\rangle_n$ [RC18, Ma21]. Should the resulting value s be negative, the *correction stage* will afterwards add p to obtain the reduction result. This can be achieved by performing an addition conditioned on the $n+1$st qubit containing the sign of the result. After this correction, the system is in the state $|s_{n+1}\rangle |\phi(xc \bmod p)\rangle_n |u\rangle_n$.

It finally only remains to uncompute the values $|s_{n+1}\rangle$ and $|u\rangle$. For this task, Rines and Chuang [RC18] propose restoring the least significant bit of $|s\rangle$ in the register that holds the sign qubit $|s_{n+1}\rangle$. This can be achieved by performing a *CNOT* operation conditioned on the least significant qubit of $|\phi(xc \bmod p)\rangle$. To illustrate why this is the case, first assume $|s_{n+1}\rangle = |0\rangle$ and therefore no correction with p has been performed previously. Here, the *CNOT* operation simply copies the basis state present in the least significant qubit of the result, which was also the least significant qubit of $|s\rangle$ since no correction was performed. In the case when $|s_{n+1}\rangle = |1\rangle$ an addition with the odd integer p is performed, which causes the least significant qubit to flip. However, since $|s_{n+1}\rangle$ starts in the flipped state, the *CNOT* operation also suffices.

This new qubit is again treated as the new most significant qubit of $|u\rangle$ as was done during the estimation stage. Therefore it only remains to uncompute $|u\rangle_{n+1} = |tp^{-1} \bmod 2^{n+1}\rangle_{n+1}$. Since t and in turn u can be expressed using the multiplication that was performed at the beginning of the algorithm as

$$u = \sum_{i=0}^{n-1} x_i (c2^n \cdot 2^i \bmod p) p^{-1} \bmod 2^{n+1},$$

it is apparent that this register can be restored to $|0\rangle$ by performing n conditional subtractions with $(c2^n \cdot 2^i \bmod p)p^{-1}$. By extensions similar to the ones presented in Section 2.4 this circuit can further be used to perform in-place multiplication.

The main improvement in the depth of this circuit stems from the fact that the relatively expensive QFT only has to be performed at the beginning and at the end of the circuit (which in some instances can also be omitted) and at the beginning of the correction stage when the most significant qubit is extracted. This contrasts the approach presented in previous sections, where the QFT is repeated for every modular addition that is performed.

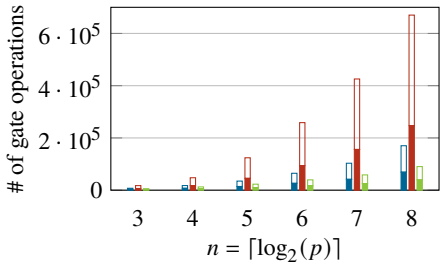

Fig. 2: The number of gate operations applied for each of the implementations presented in Section 2 for different input sizes n. The solid bars represent the share of *CNOT* operations in the total gate count. The QFT-based implementation by Beauregard [Be03] is shown in blue, the bitwise adder-based implementation by Häner et al. [HRS17] is shown in red and the Montgomery multiplication by Rines and Chuang [RC18] is shown in green.

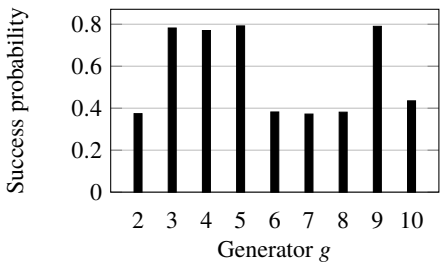

Fig. 3: The average success probability for multiple inputs to the full algorithm for the DLP for different generators and the prime $p = 11$. The upper bound for the obtainable success probability of $\phi(r)/r$ with $r = |\langle g \rangle|$ is clearly visible.

3 Results and comparison

We implemented the different circuits for modular exponentiation as well as optimisations for the overall phase estimation algorithm as they are presented in [Ma21] in Qiskit. The circuits and the overall implementation are available online[5].

Modular exponentiation gate	# of qubits	Gate operations $T(n)$
Beauregard [Be03]	$2n + 2$	$O(n^4)$
Häner, Roetteler and Svore [HRS17]	$2n + 1$	$O(n^3 \log(n))$
Rines and Chuang [RC18]	$3n + 1$	$O(n^3)$

Tab. 1: The number of required qubits excluding control qubits and the gate complexity for the modular exponentiation circuits implemented in [Ma21].

The complete algorithm for computing the discrete logarithm has the asymptotic gate complexity given in Equation (1), with $T(n)$ as listed in Table 1.

First, we compare the asymptotic gate complexity of the different modular exponentiation circuits presented in Section 2. Table 1 shows that the circuit based on Montgomery modular multiplication should perform best with respect to the other implemented circuits. It does, however, also use the most ancilla qubits, which is highlighted by its increased space complexity in Table 1.

[5] `https://github.com/mhinkie/ShorDiscreteLog`

Using the concrete implementations of these circuits, it is now also possible to compare the actual number of operations that are applied for small examples (see Figure 2). To obtain a realistic comparison for current quantum computer architectures, we compile the circuits to the gate set $\{RZ, I, \sqrt{X}, X, CNOT\}$ as this is currently used in IBM's quantum systems[6]. Furthermore, we compile for the unrestricted qubit layout of Qiskit's simulator to not introduce architecture-dependent swap operations arising from varying physical qubit layouts.

Using our implementations it is now possible to calculate discrete logarithms on quantum computers supporting Qiskit's backend. The noise-free results for one specific example are shown in Figure 3 and highlight the dependence of the success probability on the order r of the generator.

4 Summary and conclusion

In addition to the experimental results, our circuit implementations and the code to execute example instances on a simulator are available publicly to enable further research. The comparison of these gates shows that although the bitwise addition circuit by Häner et al. [HRS17] outperforms the QFT-based multiplication circuit by Beauregard [Be03] for small examples, the latter is preferable since the QFT cost is not that significant for small n. Of the three studied implementations, the most efficient in the number of gate operations for large and for small n is the circuit presented by Rines and Chuang [RC18]. Its only drawback is the comparatively large number of required qubits. In future work, the implementations provided in this work can be used on real quantum systems to further examine Shor's algorithm, its applicability to the DLP and further examine the impact of gate errors on the measurement results.

Bibliography

[Be03] Beauregard, S.: Circuit for Shor's algorithm using 2n+3 qubits. Quantum Inf. Comput., 3(2):175–185, May 2003.

[Dr00] Draper, T. G.: Addition on a Quantum Computer. arXiv preprint quant-ph/0008033, August 2000.

[GE21] Gidney, C.; Ekerå, M.: How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. Quantum, 5:433, April 2021.

[HRS17] Häner, T.; Roetteler, M.; Svore, K. M.: Factoring using 2n+2 qubits with Toffoli based modular multiplication. Quantum Inf. Comput., 17(7 & 8), June 2017.

[Ka17] Kaliski Jr., B. S.: A Quantum "Magic Box" for the Discrete Logarithm Problem. Cryptology ePrint Archive, Report 2017/745, 2017. https://ia.cr/2017/745.

[6] https://quantum-computing.ibm.com/services?services=systems&system=ibmq_santiago

[Ma21] Mandl, A.: Quantum Algorithms for the Discrete Logarithm Problem. Master's thesis, TU Wien, 2021.

[ME98] Mosca, M.; Ekert, A.: The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer. In: QCQC'98, Selected Papers. volume 1509 of LNCS. Springer, pp. 174–188, 1998.

[Mo85] Montgomery, P. L.: Modular multiplication without trial division. Mathematics of Computation, 44:519–521, 1985.

[PZ03] Proos, J.; Zalka, C.: Shor's discrete logarithm quantum algorithm for elliptic curves. Quantum Inf. Comput., 3(4):317–344, January 2003.

[RC18] Rines, R.; Chuang, I.: High Performance Quantum Modular Multipliers. arXiv preprint arXiv:1801.01081, January 2018.

[Sh97] Shor, P. W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Comp., 26(5), October 1997.

[TTK10] Takahashi, Y.; Tani, S.; Kunihiro, N.: Quantum Addition Circuits and Unbounded Fan-Out. Quantum Inf. Comput., 10(9):872–890, September 2010.

[Za98] Zalka, C.: Fast versions of Shor's quantum factoring algorithm. arXiv preprint quant-ph/9806084, 1998.