Space-Efficient and Noise-Robust Quantum Factoring

Seyoon Ragavan
MIT
sragavan@csail.mit.edu

Vinod Vaikuntanathan
MIT
vinodv@csail.mit.edu

February 15, 2024

Abstract

We provide two improvements to Regev's recent quantum factoring algorithm (arXiv:2308.06572), addressing its space efficiency and its noise-tolerance.

Our first contribution is to improve the quantum space efficiency of Regev's algorithm while keeping the circuit size the same. Our main result constructs a quantum factoring circuit using $O(n \log n)$ qubits and $O(n^{3/2} \log n)$ gates. We achieve the best of Shor and Regev (upto a logarithmic factor in the space complexity): on the one hand, Regev's circuit requires $O(n^{3/2})$ qubits and $O(n^{3/2} \log n)$ gates, while Shor's circuit requires $O(n^2 \log n)$ gates but only $O(n)$ qubits. As with Regev, to factor an n-bit integer N, we run our circuit independently $\approx \sqrt{n}$ times and applies Regev's classical postprocessing procedure.

Our optimization is achieved by implementing efficient and reversible exponentiation with Fibonacci numbers in the exponent, rather than the usual powers of 2, adapting work by Kaliski (arXiv:1711.02491) from the classical reversible setting to the quantum setting. This technique also allows us to perform quantum modular exponentiation that is efficient in both space and size without requiring significant precomputation, a result that may be useful for other quantum algorithms. A key ingredient of our exponentiation implementation is an efficient circuit for a function resembling *in-place* quantum-quantum modular multiplication. This implementation works with only black-box access to any quantum circuit for *out-of-place* modular multiplication, which we believe is yet another result of potentially broader interest.

Our second contribution is to show that Regev's classical postprocessing procedure can be modified to tolerate a constant fraction of the quantum circuit runs being corrupted by errors. In contrast, Regev's analysis of his classical postprocessing procedure requires all $\approx \sqrt{n}$ runs to be successful. In a nutshell, we achieve this using lattice reduction techniques to detect and filter out corrupt samples.

1 Introduction

Shor's landmark result from 1994 [Sho97] showed us how to factor numbers in quantum polynomial time. In particular, he constructed an $O(n^2 \log n)$-size quantum circuit that uses $O(n \log n)$ qubits (including ancilla qubits), such that $O(1)$ runs of his circuit followed by polynomial-time classical postprocessing suffices to factor an n-bit integer. A number of followup works [BCDP96, VBE96,

Sei01, Cop02, CW00, Bea03, TK06, EH17, Gid17, HRS17, Gid19, GE21] improved the Shor circuit in several ways, for example by constructing a log-depth implementation [CW00] and by reducing the number of ancilla qubits to a mere $2n+1$ [Gid17] (albeit with a larger number of gates, $\widetilde{\Theta}(n^5)$ and $\widetilde{\Theta}(n^3)$ respectively).

The size of the factoring circuit stood steady at $O(n^2 \log n)$ for nearly three decades, until Regev [Reg23] recently demonstrated an $O(n^{1.5} \log n)$-size quantum circuit for factoring. Regev shows how to factor n-bit integers using $\approx \sqrt{n}$ (parallel) runs of his factoring circuit. While the total number of operations remains the same, the hope is that the smaller Regev circuit (or a future modification of it) is easier to build and that it resists decoherence noise better.

However, Regev's factoring algorithm (that is, his quantum circuit together with the classical postprocessing algorithm) has two key limitations, the first being its space (in)efficiency and the second its noise (in)tolerance.

In a nutshell, our work provides algorithms addressing both of these problems. Both are simple and independent plug-and-play modifications to Regev's algorithm [Reg23]. We reduce the space complexity to $O(n \log n)$ by modifying the modular arithmetic part of Regev's quantum circuit, and we improve the tolerance to quantum errors by only modifying Regev's classical postprocessing procedure. We next discuss each of these problems in some more detail, and then outline our techniques and results.

Space Complexity. A key efficiency consideration in the construction of quantum circuits is their space complexity, namely, the number of qubits including the ancilla qubits used by the circuit. (For more discussion on the importance of reducing space complexity in certain quantum computing architectures, e.g. superconducting architectures, see [GE21, Gid23].) The number of qubits used in the original Shor circuit was $O(n \log n)$. Several works [BCDP96, VBE96, Bea03, TK06, Gid17, HRS17, Gid19] optimized the space complexity down to $O(n)$ qubits (and in fact, even $2n+1$) albeit at the cost of a larger circuit. However, none of these optimizations seems to apply to Regev's circuit whose space complexity stands at $O(n^{1.5})$.

To explain why, we start by reminding the reader that a key step in Shor's algorithm is the computation of the map $|x\rangle \mapsto |a^x \bmod N\rangle$ in superposition, where a is a fixed base. The fact that a is fixed allows one to precompute its powers $a, a^2, \ldots, a^{2^j}, \ldots$ classically. Once this is done, exponentiation in superposition can be done with space $O(n \log n)$ using fast multiplication [HvdH21] and *in-place* exponentiation by multiplying together the appropriate subset of the precomputed values. However, using this technique appears to prevent us from improving the size of the circuit beyond the $O(n^2)$ barrier essentially realized by Shor's algorithm; in the case of Regev's algorithm, we would have to precompute $a_i^{2^j} \bmod N$ for $\Theta(\sqrt{n})$ values of i and $\Theta(\sqrt{n})$ values of j. This is hence $\Theta(\sqrt{n} \cdot \sqrt{n} \cdot n) = \Theta(n^2)$ bits of precomputed information; a quantum circuit using all of these values would hence be expected to require $\Omega(n^2)$ gates.

Instead, if one tries to implement the map $|x\rangle |a\rangle \mapsto |x\rangle |a^x \bmod N\rangle$ via the classic square-and-multiply algorithm, one runs into the conundrum of implementing the squaring circuit modulo N, as observed by Ekerå and Gidney [Gid23]. On the one hand, doing this reversibly and in-place seems as hard as factoring N. On the other hand, writing the squared result to a new register consumes extra space; indeed, this is the source of the $O(n^{3/2})$ space requirement for Regev's circuit

[Reg23]. This state of affairs raises a natural question:

Can we achieve the best of Shor and Regev?

Concretely, can we construct a quantum factoring circuit with $\tilde{O}(n^{1.5})$ gates and $O(n)$ ancilla qubits?

Error Tolerance. Quantum decoherence noise is a central and ubiquitous obstacle to realizing quantum circuits in practice that achieve functionality that we cannot already perform classically [AAB+19, GE21, CCHL22, Cai23]. For the specific case of factoring, it appears that the associated error correction procedures are both necessary and costly. Cai [Cai23] shows that Shor's algorithm breaks down in the presence of uncorrected noise, even if one restricts to very small and structured noise in only the final QFT part of the circuit. Additionally, Gidney and Ekerå [GE21] analyze the concrete cost of factoring 2048-bit integers using Shor's algorithm while correcting for quantum errors throughout, and show that even for carefully chosen parameters, this would take 8 hours with 20 million physical qubits, a significant overhead relative to the costs of running Shor's circuit in the absence of errors.

How much error correction does one need to factor? In the case of Shor's algorithm [Sho94], a circuit with $\tilde{O}(n^2)$ gates is run $O(1)$ times. For all of these runs to work, the error probability per logical gate would have to be brought down to $\tilde{O}(1/n^2)$. While it initially appears that the situation with Regev [Reg23] would be better because of the smaller circuit, this is actually not the case upon a closer look. Indeed, Regev's circuit comprises of $\tilde{O}(n^{1.5})$ gates and is run $O(n^{0.5})$ times. Regev's analysis of his classical postprocessing only applies in the case where *every* run of the circuit is successful. Hence the logical error probability per logical gate would still have to be the same $\tilde{O}(1/n^2)$. The natural question that arises is:

Can we achieve better noise-tolerance than both *Shor* and *Regev?*

Indeed, there is a natural opportunity to improve the noise-tolerance in the case of Regev's algorithm [Reg23]: if only the classical postprocessing could be modified to tolerate a constant fraction of unsuccessful runs, then the per-gate error threshold would only have to be $\tilde{O}(1/n^{1.5})$, improving on both Shor's and Regev's algorithms.

We now proceed to describe our results for quantum factoring in more detail.

1.1 Our Results for Optimizing Space

Regev's algorithm [Reg23] achieves a smaller circuit of $O(n^{1.5}\log n)$ gates in comparison to Shor's $O(n^2 \log n)$ [Sho94]; however, for reasons discussed earlier it requires at least $\approx 3n^{1.5}$ qubits[1] as opposed to the $\tilde{O}(n)$ qubits in Shor and optimized implementations of it. In this work, we construct a quantum circuit that asymptotically achieves the best of both worlds: it has a size of $O(n^{1.5}\log n)$ matching Regev's circuit; and it requires $O(n \log n)$ qubits, nearly matching what is known for optimized versions of Shor's circuit. We compute the concrete number of qubits required

[1] In more detail, using the notation in Section 2, Regev's circuit requires at least $\log_2 D \cdot n \approx An^{1.5}$ qubits. The calculation in Appendix F.1 together with observations in Section 3.3 implies that we should take $A = 3 + o(1)$.

Algorithm	Mult. algorithm	Number of qubits	Circuit size
Shor [Sho97]	[HvdH21]	$O(n \log n)$	$O(n^2 \log n)$
Shor	Schoolbook	$\boldsymbol{O(n)}$	$O(n^3)$
Shor	[Gid19]	$\boldsymbol{O(n)}$	$O(n^{\log_2 3+1})$
Optimized Shor [Bea03, TK06, Gid17, HRS17]	Schoolbook	$\boldsymbol{2n + O(1)}$	$\widetilde{O}(n^3)$
Regev's algorithm [Reg23]	[HvdH21]	$O(n^{3/2})$	$\boldsymbol{O(n^{3/2} \log n)}$
Our optimization of Regev	[HvdH21]	$O(n \log n)$	$\boldsymbol{O(n^{3/2} \log n)}$
Our optimization of Regev	[Gid19]	$\boldsymbol{O(n)}$	$O(n^{\log_2 3+1/2})$
Our optimization of Regev	Schoolbook [RNSL17]	$\boldsymbol{(10.32 + o(1))n}$	$O(n^{5/2} \log n)$

Table 1: Comparison of our results with previous work. Asymptotically best-known results for either space or size are highlighted in bold. Note, importantly, that all values here are just for *one run of the circuit*. They do not account for the fact that a circuit for Shor's algorithm only requires $O(1)$ independent runs, while a circuit for Regev's algorithm as well as ours requires $\sqrt{n}+4$ independent runs. The number of qubits in the last line is derived from Corollary 1.4 assuming that the constant C in Regev's number-theoretic assumption is $1 + o(1)$.

(see our Theorem 1.1 below) which, using schoolbook multiplication, leads us to $(10.32 + o(1))n$ qubits and $\widetilde{O}(n^{2.5})$ gates, vis-a-vis the results of [BCDP96, VBE96, Bea03, TK06, Gid17, HRS17] which use $2n + 1$ qubits and $\widetilde{O}(n^3)$ gates. Our space improvement on Regev's circuit [Reg23] is substantial not only asymptotically but also for cryptographically relevant problem sizes; when $n = 2048$, our circuit uses at least $\approx 13\times$ fewer qubits than Regev's circuit. (See Table 1 for a full list of our results and comparison to prior work.)

As discussed earlier, approaches to modular exponentiation via precomputation or repeated squaring do not appear to yield the best-of-both-worlds results we seek. We avoid these issues with the key technique of Fibonacci exponentiation, a method of exponentiation that avoids modular squaring and instead relies solely on modular multiplication. Previous works have considered using Fibonacci numbers rather than powers of two in the exponent for particular applications [BMT+07, Kle08, Mel07]. Additionally, Kaliski [Kal17b] explicitly shows how to use this Fibonacci technique to achieve efficient modular exponentiation in the setting of *classical* reversible computation.

We show that these ideas can be adapted to the quantum setting (see Section 1.3 for a more detailed comparison of our work with [Kal17b]). While Fibonacci exponentiation seems to not be of much use in Shor's circuit, it turns out to be quite powerful in optimizing the space usage of Regev's circuit. As noted by Kaliski [Kal17b], it also offers a general way to efficiently compute modular exponents on quantum computers in cases where precomputation is impossible or inefficient; we discuss this in Appendix D.

We begin by stating our main theorem for qubit complexity, and compare the result to Regev [Reg23] and optimizations of Shor's algorithm [Sho97, BCDP96, VBE96, Bea03, HRS17, TK06, Gid17, Gid19]. In a nutshell, the algorithm by [Reg23] requires $O(n^{3/2})$ qubits, which we bring down to $O(n \log n)$ while retaining the $O(n^{3/2} \log n)$ quantum circuit size. Optimizations of Shor's algo-

rithm that retain the $O(n^2 \log n)$ circuit size require $O(n \log n)$ qubits to the best of our knowledge; thus, compared to these results, our algorithm works with the same number of qubits but achieves a $O(\sqrt{n})$ factor saving in the circuit size (just as [Reg23] does).

Main Theorem 1.1. *Assume there is a quantum circuit that implements the operation*

$$|a\rangle |b\rangle |t\rangle |0^S\rangle \mapsto |a\rangle |b\rangle |(t + ab) \bmod N\rangle |0^S\rangle$$

with G gates where N, a, b, t are all n-bit integers with $0 \leq a, b, t < N$ and $2^{n-1} \leq N < 2^n$, and S here is the number of ancilla qubits.[2]

Under conjecture 3.1 (a number-theoretic assumption proposed by [Reg23]), there exists a classical polynomial-time algorithm that outputs a non-trivial factor of N using $\sqrt{n} + 4$ calls to a quantum circuit on

$$S + \left(\frac{C+2}{\log \phi} + 6 + o(1) \right) n \approx S + (1.44C + 8.88) \cdot n$$

qubits using $O(n^{1/2} \cdot G + n^{3/2})$ gates, where $\phi = (1 + \sqrt{5})/2$ is the golden ratio. Here, C is the absolute constant from conjecture 3.1.

We now state three corollaries of this general theorem by plugging in different known results for integer multiplication. Some of these multiplication results do not have the specific structure Theorem 1.1 requires e.g. they may be classical rather than quantum, or multiply integers over $\mathbb{Z}$ rather than modulo N. However, it turns out that all these can be readily adapted to our setting; we discuss this in detail in Appendix A.

Using the classical $O(n \log n)$ multiplication algorithm due to [HvdH21] allows us to set $G = S = O(n \log n)$ and obtain the following corollary:

Corollary 1.2. *Under the same assumption as in Theorem 1.1, there is a quantum circuit for factoring that uses $O(n \log n)$ qubits and $O(n^{3/2} \log n)$ gates.*

If we want to push the number of qubits needed all the way down to $O(n)$, we can employ the space-efficient quantum implementation of Karatsuba's algorithm [KO62] due to Gidney [Gid19] which uses $S = O(n)$ ancilla qubits and $G = O(n^{\log_2 3})$ gates, yielding the following corollary:

Corollary 1.3. *Under the same assumption as in Theorem 1.1, there is a quantum circuit for factoring that uses $O(n)$ qubits and $O(n^{\log_2 3 + 1/2})$ gates.*

This latter result achieves the same space complexity as Gidney's version of Shor's algorithm [Gid19], except with $O(\sqrt{n})$ factor smaller number of gates.

Finally, if we want to use schoolbook multiplication because of its efficiency for integers of practical length, we can slightly modify the space-efficient quantum implementation of schoolbook multiplication due to [RNSL17] (see Appendix A for details about our modification). This algorithm uses $S = 2$ ancilla qubits and $G = O(n^2 \log n)$ gates, yielding the following corollary:

[2] We do not make any assumptions about the circuit's behavior on inputs where any of a, b, t are $\geq N$.

Corollary 1.4. *Under the same assumption as in Theorem 1.1, there is a quantum circuit for factoring that uses*

$$\left(\frac{C+2}{\log \phi} + 6 + o(1)\right) n \approx (1.44C + 8.88) \cdot n$$

qubits and $O(n^{5/2} \log n)$ gates, where C is the constant from Conjecture 3.1.

We compare these results with previous work on quantum circuits for factoring in Table 1. We also detail the various sources of space cost in our algorithm and potential areas for further optimizations in Section 5.4.

As we discuss in Section 3.3, following a heuristic argument by Regev [Reg23] suggests that taking $C = 1 + \epsilon$ should be sufficient for conjecture 3.1 to hold. Plugging this into the statement of Theorem 1.1 tells us that the space complexity of our circuit should be $\approx S + (10.32 + o(1))n$.

1.2 Our Results on Error-Correction

We show that Regev's classical postprocessing [Reg23] can be modified to tolerate a constant fraction of unsuccessful runs of the quantum circuit, thus only requiring a $\tilde{O}(1/n^{1.5})$ bound on the probability of logical error per logical gate.

Main Theorem 1.5. *(Informal, see Section 4.2 and Theorem 4.5 for formal statement) Assume the probability of error in one run of Regev's circuit is a sufficiently small constant $p > 0$. Then there exists a classical polynomial-time algorithm that, given $m = \Omega(\sqrt{n})$ potentially corrupt samples from Regev's quantum circuit, outputs a non-trivial factor of N with probability $\Omega(1)$.*

Our algorithm is conceptually simple. As we will explain in Section 3, uncorrupted samples from Regev's circuit will essentially be random samples from $\mathcal{L}^*/\mathbb{Z}^d$, where $\mathcal{L}$ is a lattice depending on the integer N that needs to be factored. For simplicity, assume that corrupted samples are uniform from the torus $\mathbb{R}^d/\mathbb{Z}^d$. Then the main idea is that a small linear combination of samples that includes even one corrupt sample will also be uniform and hence should not be close to the dual lattice. Based on this, we devise a filtering procedure using lattice reduction techniques to detect and filter out corrupt samples. This gives us a collection of uncorrupted samples, which can now be plugged into Regev's classical postprocessing procedure [Reg23] as-is.

However, we also extend our results beyond the case where the corrupt samples are uniform over $\mathbb{R}^d/\mathbb{Z}^d$. We define a formal model in terms of a general error distribution $\mathcal{D}$ and clearly state a general condition on $\mathcal{D}$ that suffices for our filtering procedure to go through. We defer the formal models and statements to Section 4.2.

1.3 On Kaliski's Work and Quantum-Quantum Multiplication

We compare our techniques to achieve our space-complexity result with those of Kaliski [Kal17b].

At its core, the beautiful work of Kaliski [Kal17b] constructs classical reversible and space-efficient algorithms for modular exponentiation i.e. the map $(1, a, z) \mapsto (1, a, z, a^z \bmod N)$. (Kaliski

also then shows how to "forget" a, thereby obtaining $(1, 1, z, a^z \bmod N)$, but this is neither useful nor applicable to our situation because these additional results assume the algorithm knows the order $\varphi(N)$ of the group, which we do not.)

While Kaliski's algorithm is reversible in the classical world, it is not directly applicable to quantum computation. The key arithmetic operation used by Kaliski is the mapping $(a, b) \mapsto (a, ab \bmod N)$. This is classically reversible assuming $\gcd(a, N) = 1$, but it is not trivial to implement this on a quantum computer.[3] Indeed, it is at least as difficult as inversion modulo N: the inverse of a circuit computing this mapping would map $(a, b) \mapsto (a, a^{-1}b \bmod N)$. Taking $b = 1$ would immediately yield a circuit for modular inversion, which we currently only know how to implement using $O(n^2)$ gates via the extended Euclidean algorithm [PZ03].

This problem is often referred to as *in-place quantum-quantum modular multiplication* [RC18]. "In-place" specifies that the output $ab \bmod N$ should overwrite the input register b, and "quantum-quantum" specifies that a and b are both in superposition. In contrast, "quantum-classical" multiplication would be when a is a classical constant. This is a more straightforward task, and was addressed in Shor's original paper [Sho97].

Previous work by Rines and Chuang [RC18] on in-place quantum-quantum modular multiplication constructs a very careful white-box implementation of schoolbook multiplication using Montgomery multipliers. This implementation uses $O(n^2)$ gates and $O(n)$ qubits. However, their method does not appear to be adaptable to arbitrary multiplication algorithms, and in particular, does not match state-of-the-art classical algorithms using $\widetilde{O}(n)$ gates [HvdH21, SS71].

Our algorithm approaches this problem by *sidestepping* the cost of inversion modulo N. We start by replacing each register a in Kaliski's procedure with the tuple $(a, a^{-1} \bmod N)$, so that our goal is now to implement the map $(a, a^{-1} \bmod N, b, b^{-1} \bmod N) \mapsto (a, a^{-1} \bmod N, ab \bmod N, (ab)^{-1} \bmod N)$. In Lemma 5.2, we show that this map can be implemented using $O(1)$ black-box calls to *any* quantum circuit for *out-of-place* quantum-quantum modular multiplication. This allows us to utilize any classical multiplication algorithm in our quantum circuit, including those by [HvdH21] or [SS71] (after applying the necessary procedures in Appendix A).

On the Work of [EG24]. Subsequent to a public posting of our work containing only our space complexity result, Ekerå and Gärtner [EG24] showed how to adapt both Regev's algorithm [Reg23] and our space optimization of it to obtain quantum circuits for the discrete logarithm problem over $\mathbb{Z}_p$ that achieve an analogous improvement over Shor [Sho97]. Second, concurrent to our results on error-tolerance, they also address the question of improving the error tolerance of Regev, but in a different way; instead of first detecting and filtering out corrupt samples like we do, they show under a different assumption on the distribution of corrupt samples that Regev's classical postprocessing [Reg23] as-is can withstand a constant fraction of samples being corrupted.

We note that our result described in Section 4.2 on detecting errors also readily adapts to the discrete logarithm algorithm by [EG24]. We provide an outline in Appendix E.

[3] We note that Kaliski does consider applications of his Fibonacci exponentiation idea to quantum computation [Kal17a, Kal17b], but does not appear to identify or address the problem of implementing $(a, b) \mapsto (a, ab \bmod N)$ on a quantum computer.

Organization of the Paper. We start with an overview of Regev's factoring algorithm in Section 3. We follow this up with a formal statement of our results in Section 4, the space-efficiency result in Section 5, and the error-tolerance result in Section 6.

2 Setup and Notation

We retain all notation from [Reg23] and restate it here for convenience. Let $N < 2^n$ be an n-bit number. Let $d = \lfloor \sqrt{n} \rfloor$ and $b_1, \ldots, b_d$ be some small $O(\log n)$-bit integers (e.g. b_i is the ith prime number) and let $a_i = b_i^2 \bmod N$. For any integer t, let $[t]$ denote the set $\{1, 2, \ldots, t\}$. We use $\log$ to denote the base-2 logarithm throughout this paper. Also, let ϕ denote the golden ratio. Regev's algorithm uses a number of parameters:

- Let $C > 0$ be an absolute constant given by conjecture 3.1;
- Let $A > C$ be another constant we specify later. For Regev's factoring algorithm and our space optimization, we will take $A = C + 2 + o(1)$, but a larger constant A may be needed in order to be compatible with our error detection results (see Section 4.2 for details);
- Let $R = 2^{(A+o(1))\sqrt{n}}$; and
- Let D be a power of 2 in $[2\sqrt{d} \cdot R, 4\sqrt{d} \cdot R]$. Note that D is also $2^{(A+o(1))\sqrt{n}}$. These are the same parameters R and D defined by [Reg23].

Regev defines the following lattices in d dimensions:

$$\mathcal{L} = \left\{ (z_1, \ldots, z_d) \in \mathbb{Z}^d : \prod_{i=1}^{d} a_i^{z_i} \equiv 1 \bmod N \right\}, \text{ and}$$

$$\mathcal{L}_0 = \left\{ (z_1, \ldots, z_d) \in \mathbb{Z}^d : \prod_{i=1}^{d} b_i^{z_i} \equiv \pm 1 \bmod N \right\} \subseteq \mathcal{L}.$$

We will also work closely with the dual lattice

$$\mathcal{L}^* = \left\{ y \in \mathbb{R}^d : \langle x, y \rangle \in \mathbb{Z} \ \forall x \in \mathcal{L} \right\} \supseteq \mathbb{Z}^d.$$

3 Overview of Regev's Factoring Algorithm

Regev's algorithm [Reg23] starts by ruling out simple possibilities where factoring is easy i.e. if N is even, a prime power, or shares a common factor with any of the b_i. For the remainder of this paper, we assume none of these is the case.

The goal of Regev's algorithm [Reg23] is to find one vector $z = (z_1, \ldots, z_d)$ in $\mathcal{L} \backslash \mathcal{L}_0$. Given such a vector, one can construct $b = \prod_{i=1}^{d} b_i^{z_i} \bmod N$. Since $z \in \mathcal{L}$, b must be a square root of 1 modulo N, moreover it must be a nontrivial square root of 1 since $z \notin \mathcal{L}_0$. Hence N divides $(b-1)(b+1)$ but not either term individually, so $\gcd(b-1, N)$ will be a nontrivial divisor of N.

Regev's algorithm can be broken down into two distinct pieces. First, there is a quantum circuit that essentially produces one noisy sample from $\mathcal{L}^*/\mathbb{Z}^d$. This circuit is run $d+4$ times. Secondly,

there is a classical postprocessing procedure that takes these samples and essentially recovers a basis for all vectors in $\mathcal{L}$ of norm at most $T = 2^{C\sqrt{n}}$. Under a heuristic assumption made by [Reg23], at least one of these vectors will belong to $\mathcal{L}\backslash\mathcal{L}_0$.

3.1 Overview of Regev's Quantum Circuit

We give an overview of Regev's quantum factoring circuit, just enough to understand our results; for more details, we refer the reader to the original work [Reg23]. To help keep track of space usage, all lemmas will clearly specify the number of ancilla qubits needed (if any).

3.1.1 Constructing a Superposition over Vectors z

For $s > 0$, define the Gaussian function $\rho_s : \mathbb{R}^d \to \mathbb{R}$ as

$$\rho_s(z) = \exp(-\pi||z||^2/s^2).$$

Then in this step, the algorithm constructs a discrete Gaussian state $|\psi\rangle$ within $1/\text{poly}(d)$ trace distance of the state proportional to:

$$\sum_{z\in\{-D/2,\ldots,D/2-1\}^d} \rho_R(z)\,|z\rangle\,.$$

The complexity of this step can be summarized in the following lemma from [Reg23].

Lemma 3.1. *$|\psi\rangle$ can be constructed in-place (i.e., without any ancilla qubits) with*

$$O(d(\log D + \text{poly}(\log d))) = O(n)$$

gates. The number of qubits needed to store the vector $|z\rangle$ is $d \log D = (A + o(1))n$.

Proof. We restate the outline by [Reg23] here and then explain why this can be done without ancilla qubits. We refer the reader to [GR02, Reg09] for details. Firstly, it suffices to compute a one-dimensional discrete Gaussian state over scalars $z \in \{-D/2, \ldots, D/2 - 1\}$ in-place using $O(\log D + \text{poly}(\log d))$ gates (since our d-dimensional discrete Gaussian is a tensor product of d one-dimensional discrete Gaussian states).

As explained by [Reg23], this can be achieved (up to an error of $1/\text{poly}(d)$ trace distance) by adapting a standard procedure as in [GR02, Reg09]. For the $O(\log d)$ most significant qubits, one can apply the appropriate rotation to that qubit conditioned on the values of the previous qubits, using $\text{poly}(\log d)$ gates. For the remaining qubits, the correct rotation will be very close to the $|+\rangle$ state, so we can simply apply Hadamard gates in-place to all of them. (This is where the $1/\text{poly}(d)$ trace distance error is incurred.) This takes at most $\log D$ gates.

To address the space consideration, observe that computing the $O(\log d)$ most significant qubits requires $\text{poly}(\log d)$ ancilla qubits. Once these high-order qubits have been computed, they can be copied into a fresh register (using CNOT gates), and the original circuit uncomputed. These ancilla qubits are now back in the $|0\rangle$ state, and can be *reused* as low-order qubits for the discrete Gaussian state (since we need $\approx \log D \gg \text{poly}(\log d)$ lower-order qubits). □

3.1.2 A Quantum Oracle to Compute $\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N$

Next, the algorithm computes (in superposition)

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

(the offset by $D/2$ in the exponent is to make all exponents non-negative).

Recall that we use G to denote the number of gates used by our n-bit multiplication circuit; see the statement of Theorem 1.1 for a precise description. Then, Regev achieves this step by first constructing a classical circuit with $O(\log D \cdot G) = O(n^{1/2} \cdot G)$ gates for this computation, then implementing this quantumly. For details on "compiling" classical circuits into quantum circuits, see Appendix A.1.

Regev's classical circuit uses a repeated squaring procedure, while also exploiting the fact that the a_i's are small to reduce the number of large-integer multiplications required. After compiling this into a quantum oracle and making some minor optimizations,[4] Regev obtains the following lemma:

Lemma 3.2. *As in Theorem 1.1, let G and S be the number of gates and ancilla qubits respectively for our multiplication circuit on n-bit integers. Then there exists a quantum circuit mapping*

$$|z\rangle |0^M\rangle \mapsto |z\rangle \left| \prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N \right\rangle |\psi\rangle .$$

Here,

$$M = S + O(n^{3/2})$$

is the initial number of ancilla qubits, and $|\psi\rangle$ is some possibly nonzero state on $M-n$ qubits. Moreover, this circuit uses $O(n^{1/2} \cdot G)$ gates.

We note that this step is the performance bottleneck in Regev's algorithm both in terms of gates and qubits. One of our main contributions is finding a different way to implement this oracle that only requires $S + O(n)$ qubits while retaining the same asymptotic gate complexity. We discuss our improvement in more detail in Section 4.1.

3.1.3 Measurement and QFT

The final stage of the quantum part of Regev's algorithm [Reg23] first measures the register storing $\prod_{i=1}^{d} a_i^{z_i+D/2} \pmod{N}$ to collapse the $|z\rangle$ register to a superposition over some coset from $\mathbb{Z}^d/\mathcal{L}$. Then the algorithm applies the inverse of the circuit in Lemma 3.2 to uncompute the nonzero state $|\psi\rangle$.

Now the algorithm applies an approximate QFT [Cop02] modulo D i.e. over $\mathbb{Z}_D^d$ to the $|z\rangle$ register, measures, and divides by D to obtain a vector close to a uniform sample from the dual

[4] In particular, Regev does not quite just take his classical circuit (for computing equation 1) and black-box convert it into quantum, which would result in $O(n^{3/2} \log n)$ space complexity (as explained in Appendix A.1). However, he is able to reuse the ancilla qubits used for multiplications, resulting in just $O(n^{3/2})$ qubits.

lattice $\mathcal{L}^*/\mathbb{Z}^d$. More formally, with probability $1 - 1/\operatorname{poly}(d)$, we obtain a sample of the form $w_i = v_i + \delta_i$. Here, v_i is a uniform sample from $\mathcal{L}^*/\mathbb{Z}^d$ and δ_i is some error of magnitude at most $2^{(-A+o(1))n/d}$. The sample w_i is the final output of each run of the quantum circuit.

To understand the complexity of this part of the quantum circuit, what we need is the following lemma from [Cop02, Reg23].

Lemma 3.3. *The approximate QFT can be computed in-place with circuit size*

$$O(d \log D(\log \log D + \log d)) = O(n \log n)$$

gates. The space usage here is the number of qubits needed to store $|z\rangle$, which is $d \log D = (A + o(1))n$ qubits.

3.2 Overview of Regev's Classical Postprocessing

Regev's classical postprocessing procedure [Reg23] works with the lattice $\mathcal{L}$, but it can really be viewed as an algorithm for any lattice $\Lambda \subseteq \mathbb{Z}^d$. We state Regev's result in these terms below:

Lemma 3.4. *Let $\Lambda \subseteq \mathbb{Z}^d$ be a lattice, and let $m = d+4$. Additionally, let $T > 0$ be some norm bound. Assume we are given as input m independent samples of the form*

$$w_i = v_i + \delta_i,$$

where each v_i is a uniform sample from $\Lambda^*/\mathbb{Z}^d$ and δ_i is some additive error of magnitude at most δ. Additionally, assume the following inequality:

$$(m+d)^{1/2} \cdot 2^{(m+d)/2} \cdot (m+1)^{1/2} \cdot T < \delta^{-1} \cdot (4 \det \Lambda)^{-1/m}/6.$$

Then there exists a classical polynomial-time algorithm that, with probability at least $1/4$, outputs a finite sequence of vectors $z_1, z_2, \ldots, z_l \in \Lambda$ such that any $u \in \Lambda$ with $||u||_2 \leq T$ can be written as an integer linear combination of the z_i's.

We note that the algorithm is deterministic; the success probability is taken over the randomness of the w_i's.

For the factoring algorithm, [Reg23] takes $\mathcal{L} = \Lambda$ and $T = 2^{C\sqrt{n}}$ and uses the bound that $\det \mathcal{L} \leq N < 2^n$. Plugging in and solving implies that we require $A \geq C + 2 + o(1)$ (see Appendix F.1 for this simple calculation).

We remark here that our other main contribution in this paper is extending the algorithm in Lemma 3.4 to handle cases where a small constant fraction of the m samples may be completely corrupted. Our algorithm uses Regev's classical postprocessing procedure [Reg23] as its final step, but we need to modify the analysis by [Reg23] to obtain a result slightly different from Lemma 3.4. We defer a discussion of this algorithm and its analysis to Appendix B.

3.3 Regev's Number-Theoretic Assumption

Regev's choice of the parameter T arises from the following heuristic number-theoretic assumption [Reg23]:

Conjecture 3.1. *There exists a vector in $\mathcal{L}\backslash\mathcal{L}_0$ of ℓ_2 norm at most $T = 2^{C\sqrt{n}}$, for some given constant $C > 0$.*

As observed by [Reg23], a simple pigeonhole principle argument shows that there exists a nonzero vector in $\mathcal{L}$ with norm at most $2^{(1+o(1))\sqrt{n}}$. Following Regev's heuristic argument suggests that taking $C = 1 + \epsilon$ should be sufficient.

Under this assumption, there exists some nonzero vector $u \in \mathcal{L}\backslash\mathcal{L}_0$ of ℓ_2 norm at most T. By Lemma 3.4, u is expressible as an integer linear combination of $z_1, z_2, \ldots, z_l$. Since $u \notin \mathcal{L}_0$, there exists some $i \in [l]$ such that $z_i \notin \mathcal{L}_0 \Rightarrow z_i \in \mathcal{L}\backslash\mathcal{L}_0$. Hence at least one of $z_1, \ldots, z_l$ is an element of $\mathcal{L}\backslash\mathcal{L}_0$, so we can simply try the aforementioned gcd calculation for each of them one at a time. This completes our overview of Regev's factoring algorithm.

4 Our Improvements to Regev's Algorithm

4.1 Reducing the Number of Qubits

Here, we formally state our space improvement to Lemma 3.2, thus yielding Theorem 1.1.

Lemma 4.1. *(Compare with Lemma 3.2) As in Theorem 1.1, let G and S be the number of gates and ancilla qubits respectively for our multiplication circuit on n-bit integers. Then there exists a quantum circuit mapping*

$$|z\rangle |0^M\rangle \mapsto |z\rangle \left| \prod_{i=1}^{d} a_i^{z_i + D/2} \bmod N \right\rangle |\psi\rangle .$$

Here,

$$M = S + \left(\frac{A}{\log \phi} - A + 6 + o(1) \right) n$$

is the initial number of ancilla qubits, and $|\psi\rangle$ is some possibly nonzero state on $M - n$ qubits. Moreover, this circuit uses $O(n^{1/2} \cdot G + n^{3/2})$ gates. Therefore, the total number of qubits used (i.e. the space usage) is

$$S + \left(\frac{A}{\log \phi} + 6 + o(1) \right) n$$

4.2 Tolerating Quantum Errors

Regev's algorithm assumes that all quantum gates and qubits exist without error in every run of the circuit. This may not be the case in practice; indeed, the difficulty of detecting and correcting quantum errors is a significant barrier to constructing quantum computers at scale [FMMC12, CKM19, GE21]. We introduce a model for such errors and state our results formally.

4.2.1 The Error Model

We model errors by assuming that an additional Hamming error is applied to each sample from the quantum circuit with constant probability. Concretely, let $\mathcal{D}$ be a noise distribution over $\mathbb{R}^d/\mathbb{Z}^d$,

$\eta \in \{0, 1\}$ an integer, and $p = \Theta(1)$ the probability of a quantum error in one run of the circuit. Instead of sampling $w_i = v_i + \delta_i$ directly, we will assume that the following corruption procedure is applied to it:

1. With probability $1 - p$, we observe the sample $w_i = v_i + \delta_i$. In this case, we say the sample is *uncorrupted*.
2. With probability p, we obtain the following *corrupted* sample:
   (a) Sample $\epsilon_i \leftarrow \mathcal{D}$. This sampling is independent for each i (this is a reasonable assumption since each run of the circuit is independent).
   (b) The sample we observe is now $w_i = \eta(v_i + \delta_i) + \epsilon_i$.

We comment on the role of the parameter η in this model:

- If $\eta = 1$, then we are dealing with additive errors: the sample we expect to see is perturbed by a sample from $\mathcal{D}$.
- If $\eta = 0$, then we are dealing with overwrite errors: we directly see a sample from $\mathcal{D}$.

Our results will hold for either choice of η, and hence either error model. We note that, in the important special case where $\mathcal{D}$ is uniform over $\mathbb{R}^d/\mathbb{Z}^d$, the two possibilities for η yield the same distribution.

4.2.2 Our Results

Let $\alpha > 0$ and $\gamma \in (0, \alpha - 1)$ be constants that we will specify later. We will run Regev's quantum circuit $m = \alpha d$ times.

Our results will need to depend in some way on the structure of the noise distribution $\mathcal{D}$, which we capture in the following definition which, in words, says that a distribution $\mathcal{D}$ is well-spread with respect to a lattice if there is no way to take small (non-zero) integer linear combinations (i.e. linear combinations with small integer coefficients) of samples from $\mathcal{D}$ that results in a point close to the lattice.

Definition 4.2. *Let* $\mathsf{dist}(\cdot, \cdot)$ *denote the distance on the torus* $\mathbb{R}^d/\mathbb{Z}^d$ *i.e. mod 1, and let* $\Lambda \subseteq \mathbb{Z}^d$ *be a lattice. Then we say* $\mathcal{D}$ *is* (α, γ, A)-well-spread with respect to *a lattice* Λ^* *if the following holds for any positive integer* $k \leq \alpha d$*: For i.i.d.* $\epsilon_1, \ldots, \epsilon_k \leftarrow \mathcal{D}$*, with probability* $1 - o(1)$*, there do not exist integers* $a_1, a_2, \ldots, a_k$ *such that:*

- $|a_i| \leq 2^{\frac{(\alpha - \gamma + 3 + o(1))n}{2d}}$ *for all* i*.*
- $a_i \neq 0$ *for at least one index* i*.*
- $\mathsf{dist}\big(\sum_{i=1}^k a_i \epsilon_i, \Lambda^*/\mathbb{Z}^d\big) \leq 2^{\frac{(-2A + \alpha - \gamma + 3 + o(1))n}{2d}}$.

Informally, there is a reasonable chance (over the choice of k *samples from* $\mathcal{D}$*) that no non-zero linear combination of the samples is very close to* $\Lambda^*/\mathbb{Z}^d$*.*

To justify this definition, we observe that this holds for the uniform distribution on the torus $\mathbb{R}^d/\mathbb{Z}^d$:

Lemma 4.3. *If $A > \frac{(\alpha+1)(\alpha-\gamma+3)+2}{2}$, the uniform distribution on $\mathbb{R}^d/\mathbb{Z}^d$ is (α, γ, A)-well-spread with respect to Λ^*.*

Proof. In a nutshell, this follows from a union bound and volume argument. We defer the details of the calculation to Appendix G. □

We can now formally state our result for detecting and handling errors in a constant fraction of runs in Regev's circuit. We first formulate this in terms of an arbitrary lattice $\Lambda \subseteq \mathbb{Z}^d$:

Lemma 4.4. *(Compare with Lemma 3.4) Let $\Lambda \subseteq \mathbb{Z}^d$ and n be a positive integer such that $\det \Lambda < 2^n$. Additionally, let $m = \alpha d$ and let $T > 0$ be some norm bound. Assume we are given potentially corrupt samples $w_1, \ldots, w_m$ constructed according to the error model in Section 4.2.1 with the following specifications:*

- *The initial vectors $v_1, \ldots, v_m$ are independent uniform samples from $\Lambda^*/\mathbb{Z}^d$.*
- *The small perturbations δ_i have magnitude at most δ.*
- *The errors ϵ_i are sampled from a noise distribution $\mathcal{D}$ on $\mathbb{R}^d/\mathbb{Z}^d$. (Recall that we see the uncorrupted sample $w_i = v_i + \delta_i$ with probability $1 - p$ and a corrupted sample $w_i = \eta(v_i + \delta_i) + \epsilon_i$ with probability p.)*

Let $\gamma > 0$ be a constant such that all of the following inequalities hold:

- *$p < 1 - (\gamma + 1)/\alpha$. Note that this implicitly requires $\gamma < \alpha - 1$.*
- *$A \geq \frac{\alpha-\gamma+3}{2} + o(1)$.*
- *$\left(\frac{e\alpha}{\alpha-\gamma}\right)^{\alpha-\gamma} \cdot 2^{-\gamma+1} < 1$.*

Additionally, assume the following "special inequality":

$$(\gamma d + d)^{1/2} \cdot 2^{(\gamma+1)d/2} \cdot (\gamma d + 1)^{1/2} \cdot T < \delta^{-1} \cdot (4 \det \Lambda)^{-1/(\gamma d)} \cdot 2^{-\alpha/\gamma}/6.$$

Also, assume $\mathcal{D}$ is (α, γ, A)-well-spread with respect to Λ^*. Then there exists a classical polynomial-time algorithm that, with probability at least $1/4 - o(1)$, outputs a finite sequence of vectors $z_1, z_2, \ldots, z_l \in \Lambda$ such that any $u \in \Lambda$ with $||u||_2 \leq T$ can be written as an integer linear combination of the z_i's.

We note that the algorithm is deterministic; the success probability is taken over the randomness of the w_i.

For example, we can select the parameters $\alpha = 7$ and $\gamma = 5.9$, which then require $p < 1/70$. We briefly comment on the role of each of the stated constraints on the parameters in Lemma 4.4:

- The first inequality is to limit the number of corrupted samples.

- The second inequality is needed for Lemma 6.3 and ultimately places a lower bound on the radius of the initial discrete Gaussian superposition one needs to construct (as in Section 3.1.1).
- The third inequality is needed in Appendix C; our algorithm will ultimately select some subset of γd samples and we need to take a union bound over all possible choices of this subset, necessitating an upper bound on $\binom{\alpha d}{\gamma d}$.
- Finally, the special inequality arises from analysis in Appendix B (based on that by [Reg23]) and places an additional lower bound on the Gaussian radius; intuitively, it says that the small additive errors δ_i that appear in both uncorrupted and corrupted samples cannot be too large.

To apply Lemma 4.4 in the specific case of Regev's factoring circuit [Reg23], we may once again take $\Lambda = \mathcal{L}$ and note that $\det \mathcal{L} < 2^n$, then take $T = 2^{C\sqrt{n}}$ and $\delta = 2^{(-A+o(1))n/d}$ and solve the special inequality. This is a straightforward calculation, which we defer to Appendix F.2. Combined with the observations in Section 3.3, this yields the following theorem.

Theorem 4.5. *Assume Conjecture 3.1, and let $\gamma > 0$ be a constant such that the hypotheses of Lemma 4.4 (except for the special inequality) hold for the lattice $\mathcal{L}$. Additionally, assume $A \geq \max(C + \frac{\gamma^2+\gamma+2}{2\gamma}, \frac{\alpha-\gamma+3}{2}) + o(1)$. Then there exists a polynomial-time algorithm that, given $m = \alpha d$ samples from Regev's quantum circuit corrupted according to the error model in Section 4.2.1, outputs a non-trivial factor of N with probability $1/4 - o(1)$.*

In the special case where $\mathcal{D}$ is the uniform distribution over $\mathbb{R}^d/\mathbb{Z}^d$, combining Theorem 4.5 with Lemma 4.3 implies that one should take

$$A \geq \max\left(C + \frac{\gamma^2+\gamma+2}{2\gamma}, \frac{(\alpha+1)(\alpha-\gamma+3)+2}{2}\right) + o(1)\ .$$

As mentioned earlier, our algorithm and analysis in Lemma 4.4 are also directly applicable to the recent adaptation by Ekerå and Gärtner [EG24] of Regev's factoring algorithm [Reg23] to the discrete logarithm problem modulo prime p. We provide an outline in Appendix E.

5 Reducing Qubits via Space-Efficient Exponentiation

In this section, we construct a quantum oracle that computes the desired mapping in Lemma 4.1.

5.1 Reversible Fibonacci Exponentiation

This is the key idea used by Kaliski [Kal17b] for space-efficient and reversible classical exponentiation; we restate it here and explain its applicability to our optimization. The bottleneck in [Reg23]'s construction of this oracle is in the use of repeated squaring. As a simple example, suppose we have some n-bit integer $|a\rangle$ and we wish to compute $|a^{2^k} \bmod N\rangle$ for some k. The natural classical-inspired approach would be to use repeated squaring as follows:

$$|a\rangle \mapsto \left|a^2 \bmod N\right\rangle \mapsto \left|a^4 \bmod N\right\rangle \mapsto \ldots \mapsto \left|a^{2^k} \bmod N\right\rangle .$$

The problem is that none of these operations are easily reversible; computing square roots mod N is hard, and there is also the fact that squaring mod N is not one-to-one. Thus carrying out this computation requires storing all of these quantities in separate registers. This leads to an $O(k)$ blow-up in the space complexity, which appears wasteful.

To get around this, we use the fact that it is much easier to implement reversible multiplication than squaring. Concretely, if we could implement an operation resembling $|a\rangle|b\rangle \mapsto |a\rangle|ab \bmod N\rangle$, we could proceed as follows:

$$\begin{aligned}|a\rangle|a\rangle \mapsto |a\rangle|a^2 \bmod N\rangle &\mapsto |a^3 \bmod N\rangle|a^2 \bmod N\rangle \\ &\mapsto |a^3 \bmod N\rangle|a^5 \bmod N\rangle \mapsto \dots\end{aligned}$$

We now find Fibonacci numbers rather than powers of 2 in the exponent. For this reason, previous works have also considered using Fibonacci numbers rather than powers of 2 in the exponent for fast and reversible exponentiation [BMT+07, Kal17b, Kle08, Mel07].

Notice that when using this idea, we will have an extra register lingering around. For example, if our goal is to compute $|a^5 \bmod N\rangle$ in one register, this would give us $|a^3 \bmod N\rangle$ in the other register. We would like to be able to clean this up if needed, but fortunately this is straightforward; we can copy the final output to a new register, and then uncompute our circuit to clean up all intermediate qubits. This will just leave us with the inputs and outputs of our computation. We will use this idea repeatedly in our algorithm.

If we could actually implement $|a\rangle|b\rangle \mapsto |a\rangle|ab \bmod N\rangle$, we could directly apply Algorithm FibExp from [Kal17b] (with a straightforward adaptation to computing a product of multiple exponents rather than one exponent). This task is known as "in-place quantum-quantum modular multiplication". (See Section 1.3 for definitions of these terms.) The issue is that this appears difficult to do on a quantum computer for arbitrary multiplication algorithms, to the best of our knowledge. Rines and Chuang [RC18] show that this operation can be done on a quantum computer in $O(n^2)$ gates with a careful implementation of schoolbook multiplication using Montgomery multipliers. However, we would like to be able to cleanly adopt and use any classical multiplication algorithm in our circuit, for example to improve asymptotic gate complexity.

To achieve this, we show how to implement an "abstract form" of in-place quantum-quantum modular multiplication in the next two lemmas, using only black-box access to a multiplication circuit. As in Theorem 1.1, we assume throughout that this multiplication circuit maps

$$|a\rangle|b\rangle|t\rangle|0^S\rangle \mapsto |a\rangle|b\rangle|(t+ab) \bmod N\rangle|0^S\rangle$$

where a, b, t are all n-bit integers and $0 \leq a, b, t < N$. S hence denotes the number of ancilla qubits used by the circuit, and we also use G to denote the number of gates in this circuit. In other words, this circuit implements *out-of-place* quantum-quantum modular multiplication.

Our lemmas will also make use of some "dirty ancilla qubits" that may not necessarily be in the $|0\rangle$ state. This will allow us to reuse qubits from other parts of our algorithm and cut down on the number of qubits needed in our circuit. The idea of using dirty ancilla qubits to achieve space savings was introduced by [HRS17] in relation to implementing Shor's algorithm with fewer qubits.

The first lemma we need is essentially due to Shor [Sho97]. Using the terminology introduced in Section 1.3, this addresses the task of in-place quantum-classical modular multiplication.

Lemma 5.1. *Let $a \in [0, N-1]$ be an integer coprime to N. Then there exists a circuit using $O(G+n)$ gates mapping*

$$|x\rangle |0^{S+n}\rangle |g\rangle \mapsto |ax \bmod N\rangle |0^{S+n}\rangle |(-a^{-1}g) \bmod N\rangle$$

for any integers x, g that are reduced mod N.

Note that this computation uses and restores $S+n$ clean ancilla qubits, while applying some reversible transformation to n dirty ancilla qubits that initially store the state $|g\rangle$.

Proof. We can classically precompute $a^{-1} \bmod N$ efficiently using the extended Euclidean algorithm. Now proceed as follows using our quantum multiplication circuit given in Theorem 1.1:

$$\begin{aligned}
&|x\rangle |0^n\rangle |g\rangle |0^S\rangle \\
&\quad \to |x\rangle |a\rangle |g\rangle |0^S\rangle \text{ (writing in a classical constant using bit-flips)} \\
&\quad \to |x\rangle |a\rangle |(g+ax) \bmod N\rangle |0^S\rangle \\
&\quad \to |x\rangle |-a^{-1} \bmod N\rangle |(g+ax) \bmod N\rangle |0^S\rangle \\
&\qquad\qquad \text{(writing in a classical constant again)} \\
&\quad \to |(x + (-a^{-1} \cdot (g+ax))) \bmod N\rangle |-a^{-1} \bmod N\rangle |(g+ax) \bmod N\rangle |0^S\rangle \\
&\qquad\qquad = |(-a^{-1}g) \bmod N\rangle |-a^{-1} \bmod N\rangle |(g+ax) \bmod N\rangle |0^S\rangle \\
&\quad \to |(-a^{-1}g) \bmod N\rangle |a\rangle |(g+ax) \bmod N\rangle |0^S\rangle \\
&\qquad\qquad \text{(writing in a classical constant again)} \\
&\quad \to |(-a^{-1}g) \bmod N\rangle |a\rangle |(g+ax+(-a^{-1}g \cdot a)) \bmod N\rangle |0^S\rangle \\
&\qquad\qquad = |(-a^{-1}g) \bmod N\rangle |a\rangle |ax \bmod N\rangle |0^S\rangle \\
&\quad \to |(-a^{-1}g) \bmod N\rangle |0^n\rangle |ax \bmod N\rangle |0^S\rangle \\
&\qquad\qquad \text{(writing in a classical constant again)} \\
&\quad \to |ax \bmod N\rangle |0^n\rangle |(-a^{-1}g) \bmod N\rangle |0^S\rangle
\end{aligned}$$

This runs our multiplication circuit three times and does $O(n)$ bit flips and bit swaps, for a total of $O(G+n)$ gates. This completes our proof. □

In the next lemma, we adapt this idea to the quantum-quantum case i.e. where a may be a superposition of integers rather than a classical constant. Note that this lemma does not exactly implement quantum-quantum modular multiplication, but rather an abstracted form of it where we view $|a\rangle |a^{-1} \bmod N\rangle$ as an abstract representation of a.

Lemma 5.2. *There exists a quantum circuit using $O(G+n)$ gates such that, for all n-bit integers $a, b, g \in [0, N-1]$ such that a and b are coprime to N, it will map*[5]

$$\begin{aligned}
&|a\rangle |a^{-1} \bmod N\rangle |b\rangle |b^{-1} \bmod N\rangle |g\rangle |0^S\rangle \\
&\qquad\qquad \mapsto |a\rangle |a^{-1} \bmod N\rangle |ab \bmod N\rangle |(ab)^{-1} \bmod N\rangle |g\rangle |0^S\rangle .
\end{aligned}$$

Note that this computation uses and restores S clean ancilla qubits and n dirty ancilla qubits.

[5] We are not concerned about the circuit's behavior on other basis states.

Proof. The quantum multiplication circuit assumed in Theorem 1.1 allows us the following computation when $0 \leq t < N$:

$$|a\rangle |b\rangle |t\rangle |0^S\rangle \mapsto |a\rangle |b\rangle |(t + ab) \bmod N\rangle |0^S\rangle .$$

We can also hence use the inverse of this circuit, which will behave as follows when $0 \leq t < N$:

$$|a\rangle |b\rangle |t\rangle |0^S\rangle \mapsto |a\rangle |b\rangle |(t - ab) \bmod N\rangle |0^S\rangle .$$

We can use these two circuits to proceed as follows. (All the computations below are $\bmod N$, which we omit for compactness. For example, when we say a^{-1}, we mean $a^{-1} \bmod N$.)

$$\begin{aligned}
&|a\rangle |a^{-1}\rangle |b\rangle |b^{-1}\rangle |g\rangle |0^S\rangle \\
&\quad \to |a\rangle |a^{-1}\rangle |b\rangle |b^{-1}\rangle |g + ab\rangle |0^S\rangle \\
&\quad \to |a\rangle |a^{-1}\rangle |b - (a^{-1} \cdot (g + ab))\rangle |b^{-1}\rangle |g + ab\rangle |0^S\rangle \\
&\qquad\quad = |a\rangle |a^{-1}\rangle |-a^{-1}g\rangle |b^{-1}\rangle |g + ab\rangle |0^S\rangle \\
&\quad \to |a\rangle |a^{-1}\rangle |-a^{-1}g\rangle |b^{-1}\rangle |(g + ab) + a \cdot (-a^{-1}g)\rangle |0^S\rangle \\
&\qquad\quad = |a\rangle |a^{-1}\rangle |-a^{-1}g\rangle |b^{-1}\rangle |ab\rangle |0^S\rangle \\
&\quad \to |a\rangle |a^{-1}\rangle |-a^{-1}g + a^{-1}b^{-1}\rangle |b^{-1}\rangle |ab\rangle |0^S\rangle \\
&\quad \to |a\rangle |a^{-1}\rangle |-a^{-1}g + a^{-1}b^{-1}\rangle |b^{-1} - (a \cdot (-a^{-1}g + a^{-1}b^{-1}))\rangle |ab\rangle |0^S\rangle \\
&\qquad\quad = |a\rangle |a^{-1}\rangle |-a^{-1}g + a^{-1}b^{-1}\rangle |g\rangle |ab\rangle |0^S\rangle \\
&\quad \to |a\rangle |a^{-1}\rangle |(-a^{-1}g + a^{-1}b^{-1}) + a^{-1} \cdot g\rangle |g\rangle |ab\rangle |0^S\rangle \\
&\qquad\quad = |a\rangle |a^{-1}\rangle |(ab)^{-1}\rangle |g\rangle |ab\rangle |0^S\rangle \\
&\quad \to |a\rangle |a^{-1}\rangle |ab\rangle |(ab)^{-1}\rangle |g\rangle |0^S\rangle .
\end{aligned}$$

This runs our multiplication circuit four times and its inverse twice and then does a constant number of swaps of n-bit registers at the end, for a total of $O(G + n)$ gates. This completes our proof. □

Before we continue, we state a variant of this lemma that will allow us to save some more qubits later on:

Corollary 5.3. *Assume there exist unitary circuits U_1, U_2 and an integer $a \in [0, N-1]$ coprime to N such that:*

1. *U_1 constructs $|a\rangle$ with G_1 gates and S_1 clean ancilla qubits, possibly using some dirty ancilla qubits. Formally, given some auxiliary information $|\phi\rangle$, it maps $|\phi\rangle |0^{S_1+n}\rangle \mapsto |\phi_1\rangle |a\rangle |0^{S_1}\rangle$. (Here, we include dirty ancilla qubits in the state $|\phi\rangle$, so at the end we may have $|\phi_1\rangle \neq |\phi\rangle$.)*

2. *U_2 constructs $|a^{-1} \bmod N\rangle$ with G_2 gates and S_2 clean ancilla qubits, possibly using some dirty ancilla qubits. Formally, it maps*

$$|\phi\rangle |0^{S_2+n}\rangle \mapsto |\phi_2\rangle |a^{-1} \bmod N\rangle |0^{S_2}\rangle \ .$$

Then there exists a circuit using $O(G + G_1 + G_2 + n)$ gates mapping

$$|\phi\rangle\,|b\rangle\,|b^{-1} \bmod N\rangle\,|g\rangle\,|0^{\max(S,S_1,S_2)+n}\rangle \\ \mapsto |\phi\rangle\,|ab\rangle\,|(ab)^{-1} \bmod N\rangle\,|g\rangle\,|0^{\max(S,S_1,S_2)+n}\rangle\,.$$

Informally, we can reuse one register to store either $|a\rangle$ or $|a^{-1} \bmod N\rangle$ to perform the same computation as in Lemma 5.2. (Note that all dirty ancilla qubits are ultimately restored to their original state.)

Proof. This follows from a very similar computation as in Lemma 5.2, except we now need to repeatedly use U_1 and U_2 to compute and uncompute $|a\rangle$ and $|a^{-1}\rangle$ into the same register. We defer details to Appendix H. □

The next lemma essentially plugs Lemma 5.2 into Kaliski's algorithm [Kal17b], while taking care to optimize space usage. We quickly set up some notation. Define the Fibonacci numbers by $F_0 = 0, F_1 = 1$, and then $F_k = F_{k-1} + F_{k-2}$ for $k \geq 2$. Let K be maximal such that $F_K \leq D$. We have:

$$K = (1 + o(1)) \cdot \frac{\log D}{\log \phi} = (1 + o(1)) \cdot \frac{(A + o(1))\sqrt{n}}{\log \phi} = (\alpha + o(1))\sqrt{n},$$

where we let $\alpha = \frac{A}{\log \phi}$.

Also, let $|\psi(a)\rangle$ denote the state $|a\rangle\,|a^{-1} \bmod N\rangle$ for an n-bit integer a coprime to and reduced modulo N. Each $|\psi(a)\rangle$ hence requires $2n$ qubits to store.

Lemma 5.4. *[Kal17b] There exists a quantum circuit using $O(K(G + n)) = O(n^{1/2}(G + n))$ gates such that, for all n-bit integers $c_1, \ldots, c_K$ coprime to and reduced modulo N, it will map*

$$|\psi(c_1)\rangle \ldots |\psi(c_K)\rangle\,|0^{S+5n}\rangle \\ \mapsto |\psi(c_1)\rangle \ldots |\psi(c_K)\rangle \left|\psi\left(\prod_{j=2}^{K} c_j^{F_{j-1}}\right)\right\rangle \left|\psi\left(\prod_{j=1}^{K} c_j^{F_j}\right)\right\rangle |0^{S+n}\rangle\,.$$

(All registers throughout this lemma and its proof work with integers mod N, so we drop the $\bmod N$ everywhere for convenience.)

Proof. We will use $4n$ of our ancilla qubits to store two states $|\psi(x_1)\rangle$ and $|\psi(x_2)\rangle$. Thus the state at any given point in our algorithm will be

$$|\psi(c_1)\rangle\,|\psi(c_2)\rangle \ldots |\psi(c_K)\rangle\,|\psi(x_1)\rangle\,|\psi(x_2)\rangle\,|0^{S+n}\rangle\,.$$

We will update the values of x_1 and x_2 throughout the algorithm.

Lemma 5.2 tells us that we can perform the operation $|\psi(x)\rangle\,|\psi(y)\rangle\,|0^{S+n}\rangle \mapsto |\psi(x)\rangle\,|\psi(xy)\rangle\,|0^{S+n}\rangle$ using $O(G+n)$ gates. We denote this as updating $y \leftarrow xy$ and will iteratively use this, as described in Algorithm 5.1.

Before we show the correctness of our algorithm, let us quickly analyze its complexity. The initialization takes $O(n)$ gates. Within the loop, there are $O(G+n)$ gates from calls to Lemma 5.2

Algorithm 5.1: Fibonacci multi-exponentiation

Data: Initial state $|\psi(c_1)\rangle\,|\psi(c_2)\rangle\ldots|\psi(c_K)\rangle\,|0^{S+5n}\rangle$

Result: Final state $|\psi(c_1)\rangle\,|\psi(c_2)\rangle\ldots|\psi(c_K)\rangle\,|\psi(\prod_{j=2}^{K}c_j^{F_{j-1}})\rangle\,|\psi(\prod_{j=1}^{K}c_j^{F_j})\rangle\,|0^{S+n}\rangle$

1. Initialize $x_1 \leftarrow 1$ and $x_2 \leftarrow 1$. This is just a matter of copying some qubits into ancilla qubits, and the state is now $|\psi(c_1)\rangle\,|\psi(c_2)\rangle\ldots|\psi(c_K)\rangle\,|\psi(x_1)\rangle\,|\psi(x_2)\rangle\,|0^{S+n}\rangle = |\psi(c_1)\rangle\,|\psi(c_2)\rangle\ldots|\psi(c_K)\rangle\,|\psi(1)\rangle\,|\psi(1)\rangle\,|0^{S+n}\rangle$.

2. Repeat the following for $j = K, K-1, \ldots, 1$ in that order:

   - Update $x_1 \leftarrow x_1x_2$ using Lemma 5.2.
   - Update $x_1 \leftarrow x_1c_j$ using Lemma 5.2.
   - Swap x_1 and x_2 (i.e. swap $|\psi(x_1)\rangle$ and $|\psi(x_2)\rangle$).

and then $O(n)$ gates to swap x_1 and x_2. The loop has K iterations, so the total number of gates is $O(K(G+n)) = O(n^{1/2}(G+n))$.

To show that the algorithm runs correctly, we claim that for all $j \le K-1$, at the end of round j, we will have $x_1 = \prod_{i=j+1}^{K} c_i^{F_{i-j}}$ and $x_2 = \prod_{i=j}^{K} c_i^{F_{i+1-j}}$. This will follow by a straightforward induction, which we present in Appendix I. □

Note, importantly, that we would not want to use Lemma 5.4 directly in our algorithm; storing $c_1, \ldots, c_K$ at the same time is an undesirable space overhead. Instead, we will compute $c_1, \ldots, c_K$ on an as-needed basis and apply the same idea; this way, we only need to store one of them at any given time. Additionally, we will leverage the optimization in Corollary 5.3 to ensure that we do not even need to store $|c_j\rangle$ and $|c_j^{-1} \bmod N\rangle$ at the same time. We explain how to do this and how we define the c_j's in the next section.

5.2 Combining Fibonacci Exponentiation with Regev's Optimization

To use the results from Section 5.1, we want to decompose each of our exponents as a sum of distinct Fibonacci numbers. Concretely, recall that we want to compute the expression $\prod_{i=1}^{d} a_i^{z_i+D/2}$. So for each $i \in [d]$, we would like to write:

$$z_i + D/2 = \sum_{j=1}^{K} z_{i,j}F_j,$$

for $z_{i,j} \in \{0,1\}$. It is well-known that this can be achieved with a simple greedy algorithm; see Appendix J for details. Our first task is to compute these coefficients:

Lemma 5.5. *There exists a quantum circuit using $O(n^{3/2})$ gates mapping the state*

$$|z\rangle\,|0^{dK-d\log D}\rangle\,|0^{O(\sqrt{n})}\rangle \mapsto |z_{i,j} : i \in [d], j \in [K]\rangle\,|0^{O(\sqrt{n})}\rangle\,.$$

Proof. We repeat the following greedy procedure for each $i \in [d]$. Note that integers here are computed in absolute terms, rather than modulo N. We need the ability to compute in-place additions and subtractions on $O(\sqrt{n})$-bit integers with $O(\sqrt{n})$ ancilla qubits; it was shown by [Dra00] that this is possible. We also need to be able to compare integers of length $O(\sqrt{n})$, but this need not be in-place so can also be done with $O(\sqrt{n})$ ancilla qubits.

1. Let t denote the number in the register currently holding z_i. First update $t \leftarrow t + D/2$ (so that this register now holds $z_i + D/2$).
2. Set aside K ancilla qubits to hold $z_{i,j}$ for $j \in [K]$, so that $z_{i,j} = 0$ for all j initially.
3. Now for each $j = K, K-1, \ldots, 1$, check whether $t \geq F_j$ and write the output of this comparison to the qubit $z_{i,j}$. Then use $z_{i,j}$ as a control qubit to conditionally update $t \leftarrow t - F_j$.
4. By Lemma J.1, this greedy algorithm will correctly decompose $z_i + D/2$ as a sum $\sum_{j=1}^{K} z_{i,j} F_j$ of distinct Fibonacci numbers, and we will have $t = 0$ at the end. Hence we have freed up those $\log D$ bits as ancilla qubits to use in later steps.

We have already observed that correctness follows from Lemma J.1. For the runtime, each step of the innermost loop over j uses $O(\log D) = O(\sqrt{n})$ gates. So, each step of the outer loop over i uses $O(K \log D) = O(n)$ gates. Finally, multiplying by $d = \sqrt{n}$ yields a gate complexity of $O(n^{3/2})$.

Finally, we address space. All individual steps in the loop can clearly be done using $O(\sqrt{n})$ ancilla qubits (which we can then reuse). Other than that, each step of the loop consumes K ancilla qubits but then frees up $\log D$ ancilla qubits. The total initial ancilla requirement is hence $d(K - \log D) + O(\sqrt{n})$ as desired. □

Now that we can calculate the $z_{i,j}$'s, let us write our desired expression in those terms:

$$\prod_{i=1}^{d} a_i^{z_i + D/2} = \prod_{i=1}^{d} \prod_{j=1}^{K} a_i^{z_{i,j} F_j} = \prod_{j=1}^{K} \left(\prod_{i=1}^{d} a_i^{z_{i,j}} \right)^{F_j}.$$

We can calculate each $\prod_{i=1}^{d} a_i^{z_{i,j}}$ efficiently following the idea by [Reg23] to exploit the fact that the a_i's are very small integers:

Lemma 5.6. *[Reg23] There exists a quantum circuit using $O(\sqrt{n} \log^3 n)$ gates mapping*

$$|t_1\rangle \ldots |t_d\rangle \, |0^{\widetilde{O}(\sqrt{n})}\rangle \mapsto |t_1\rangle \ldots |t_d\rangle \left| \prod_{i=1}^{d} a_i^{t_i} \right\rangle |0^{\widetilde{O}(\sqrt{n})}\rangle .$$

Here, the $t_i \in \{0, 1\}$ for all i.

Proof. [Reg23] shows that such a computation can be done classically with $O(d \log^3 d) = O(\sqrt{n} \log^3 n)$ gates. We can implement this using unitary gates (as explained in Appendix A.1) and just use $O(\sqrt{n} \log^3 n) = \widetilde{O}(\sqrt{n})$ ancilla qubits for all intermediate values in the circuit. Then we copy the final output $|\prod_{i=1}^{d} a_i^{t_i}\rangle$ to a fresh register; each a_i is $O(\log d)$ bits so we need $O(d \log d) = \widetilde{O}(\sqrt{n})$

ancilla qubits and $O(d \log d) = O(\sqrt{n} \log n)$ gates to do this. Finally, we can just uncompute the original circuit to restore all ancilla qubits to $|0\rangle$. In total, we have used $\widetilde{O}(\sqrt{n})$ ancilla qubits and $O(\sqrt{n} \log^3 n)$ gates, as desired. □

Combining these ideas allows us to outline the algorithm. For each $j \in [K]$, define $c_j = \prod_{i=1}^{d} a_i^{z_{i,j}}$. Then we have:

$$\prod_{i=1}^{d} a_i^{z_i + D/2} = \prod_{j=1}^{K} \left(\prod_{i=1}^{d} a_i^{z_{i,j}} \right)^{F_j} = \prod_{j=1}^{K} c_j^{F_j}.$$

We can use Lemma 5.6 to compute each c_j, and Lemma 5.4 to finally compute this product. The only missing detail is that of computing c_j^{-1}, which we do as follows:

$$c_j^{-1} = \prod_{i=1}^{d} a_i^{-z_{i,j}} = \left(\prod_{i=1}^{d} a_i^{-1} \right) \cdot \prod_{i=1}^{d} a_i^{1 - z_{i,j}}.$$

We can compute the latter expression using Lemma 5.6 since the exponents are now in $\{0, 1\}$. Then we can apply Lemma 5.1 using the classical constant $\prod_{i=1}^{d} a_i^{-1}$ to finish computing c_j^{-1}. We formally detail all of this in the next section.

5.3 Proof of Lemma 4.1

Our procedure is detailed in Algorithm 5.2. We track the use of ancilla qubits in the algorithm description.

First we address the correctness of our algorithm. We make some important observations to justify our use of dirty ancilla qubits:

- At step j in the loop, our algorithm only uses qubits $z_{i,j'}$ for $j' \neq j$ as dirty ancilla qubits, so the fact that these qubits might change value during the loop will not affect how x_1 and x_2 are updated.
- We also need to check that our dirty ancilla qubits are usable in Lemmas 5.1 and 5.2 and Corollary 5.3 i.e. that $g \in [0, N-1]$, where $|g\rangle$ is the value stored by these ancilla qubits. Initially, this is ensured by the fact that the first bit of g is 0 i.e. $g < 2^{n-1} \leq N$. Additionally, even though Lemma 5.1 modifies g, it replaces g with some other value that is reduced mod N. So g will always be in $[0, N-1]$.
- Any application of Lemma 5.2 will preserve the value of these dirty ancilla qubits.
- Any application of Corollary 5.3 will temporarily modify and then restore any dirty ancilla qubits. The modifications only happen through internal calls to Lemma 5.1, which are uncomputed within the procedure of Corollary 5.3. (Recall that the modifications to dirty ancilla qubits are captured through the changes between $|\phi\rangle, |\phi_1\rangle$, and $|\phi_2\rangle$ in the proof of Corollary 5.3.)

Algorithm 5.2: Quantum oracle for $\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N$

Data: Initial state $|z\rangle$ and $S + (\alpha - A + 6 + o(1))n$ ancilla qubits in the $|0\rangle$ state.

Result: Final state comprising $|z\rangle\,|\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N\rangle$ and $S + (\alpha - A + 5 + o(1))n$ qubits in some state (which may not be $|0\rangle$).

1. Use Lemma 5.5 to compute and store the values $|z_{i,j}\rangle$ for all $i \in [d]$ and $j \in [K]$. Note that this step also "overwrites" the qubits storing $|z\rangle$. (This consumes $dK - d\log D = (\alpha - A + o(1))n$ qubits, leaving $S + (6 + o(1))n$ ancilla qubits.)

2. Set aside $4n$ ancilla qubits. These will store our states $|\psi(x_1)\rangle$ and $|\psi(x_2)\rangle$. Initialize $x_1 \leftarrow 1$ and $x_2 \leftarrow 1$. (This leaves $S + (2 + o(1))n$ ancilla qubits.)

3. Repeat the following for $j = K, K-1, \ldots, 1$ in that order:

   (a) Consider the qubits $z_{i,j'}$ for $i \in [d]$ and $j' \neq j$. There are $d(K-1) \geq (\alpha - o(1))n \geq (\frac{2}{\log \phi} - o(1))n > 2n$ such qubits, and we will not use any of them for this iteration of the loop. Hence we may take $n-1$ of these qubits and pre-pend one clean qubit in the $|0\rangle$ state to obtain n dirty ancilla qubits. We now have $S + 2n$ clean ancilla qubits available.

   (b) Update $x_1 \leftarrow x_1 x_2$ using Lemma 5.2. (This temporarily uses and restores S clean ancilla qubits and n dirty ancilla qubits, which we have.)

   (c) We now update $x_1 \leftarrow x_1 c_j$ using Corollary 5.3. To apply this, we need to construct circuits to compute each of $|c_j\rangle$ and $|c_j^{-1}\rangle$:

   - To compute $|c_j\rangle$, use Lemma 5.6. This incurs costs $G_1 = O(\sqrt{n}\log^3 n)$ and $S_1 = \widetilde{O}(\sqrt{n})$.
   - To compute $|c_j^{-1}\rangle$, first calculate the state $|\prod_{i=1}^{d} a_i^{1-z_{i,j}} \bmod N\rangle$ using Lemma 5.6 and store it in another n-bit register. Then use Lemma 5.1 with the classical constant $\prod_{i=1}^{d} a_i^{-1} \bmod N$ to update this register to contain $|c_j^{-1} \bmod N\rangle$. This incurs total costs $G_2 = O(G + n)$ and $S_2 = S + n$.

   Hence this call to Corollary 5.3 uses $O(G+n)$ gates. It will temporarily use and restore $\max(S, S_1, S_2) + n = S + 2n$ clean ancilla qubits and n dirty ancilla qubits, which we have.

   (d) Swap x_1 and x_2 (i.e. swap the registers $|\psi(x_1)\rangle$ and $|\psi(x_2)\rangle$).

Now that we have shown that our use of dirty ancilla qubits does not affect anything, correctness follows from the proof of Lemma 5.4; at the end of step 3, we will have $|x_2\rangle = |\prod_{j=1}^{K} c_j^{F_j} \bmod N\rangle = |\prod_{i=1}^{d} a_i^{z_i+D/2} \bmod N\rangle$.

Finally, we check the size of our circuit. The steps before the loop use $O(n^{3/2})$ gates. Now, for each step of the loop over j:

1. Each application of Lemma 5.2 uses $O(G+n)$ gates.
2. Each application of Corollary 5.3 uses $O(G+n)$ gates.
3. The final swap uses $O(n)$ gates.

The loop runs for K iterations, so the total circuit size from the loop will be $O(K(G+n)) = O(n^{1/2}(G+n))$ gates. Combining this with the initialization steps yields a circuit of size $O(n^{1/2} \cdot G + n^{3/2})$, thus completing the proof of Lemma 4.1. □

5.4 Potential Future Space Optimizations

We have shown that Regev's factoring algorithm can be implemented using just

$$S + \left(\frac{A}{\log \phi} + 6 + o(1)\right)n$$

qubits, while retaining the asymptotic $O(n^{1/2} \cdot G + n^{3/2})$ circuit size. We remind the reader that here G and S denote the number of gates and the number of ancilla qubits respectively for our multiplication circuit on n-bit integers.

A natural question is whether this can be optimized further, particularly the constant $\frac{A}{\log \phi} + 6$ in the linear term. To this end, we describe the various space costs incurred by our algorithm below:

1. $\alpha n = \left(\frac{A}{\log \phi} + o(1)\right)n$ qubits are used to store the qubits $z_{i,j}$ for $i \in [d]$ and $j \in [K]$. This is a slight blow-up from the $(A + o(1))n$ bits that we really need to store all the z_i's.

   This does appear inefficient, because at any given point in our algorithm we are only interested in one particular value of j. However, the natural way to compute the $z_{i,j}$'s involves iterating through each value of i (since we decompose $z_i + D/2$ as a sum of Fibonacci numbers to obtain the $z_{i,j}$'s). A potential workaround would be to find a way to compute $z_{i,j}$ given i and j in a more efficient way than actually calculating the entire representation of $z_i + D/2$ as a sum of Fibonacci numbers, but we are not aware of how to do this.

2. $S + n$ clean ancilla qubits are needed for the multiplication operations given in Lemmas 5.1 and 5.2. This is already quite optimized due to our use of dirty ancilla qubits in these lemmas. However, under certain conditions this can be brought down to S (which is necessary for us to be able to use our multiplication circuit).

   The only reason we need the extra n ancilla qubits is in Lemma 5.1, which we use with $a = (\prod_{i=1}^{d} a_i^{-1}) \bmod N$. These n qubits are used to write in a and $-a^{-1}$ so that we can use

our multiplication circuit to multiply by these integers. Note however that a and $-a^{-1}$ are classical constants and can hence be precomputed classically.

If the quantum multiplication circuit is just an implementation of a classical multiplication circuit with unitaries (e.g. when we use the $O(n \log n)$ multiplication algorithm by [HvdH21]), then we can save n clean ancilla qubits as follows. Following ideas by Shor [Sho97], one can bake the values a and $-a^{-1}$ into the given classical circuit architecture to obtain hardwired classical circuits for "multiplication by a" and "multiplication by $-a^{-1}$". Then these hardwired classical circuits can be implemented with unitary gates. Now we no longer require a and $-a^{-1}$ to be explicitly written down in another register.

For general quantum multiplication circuits, we are not aware of a way to achieve this hardwiring.

3. $4n$ ancilla qubits are needed to store the "accumulator" states $|\psi(x_1)\rangle$ and $|\psi(x_2)\rangle$. We are not aware of any approaches to bypass this requirement.

4. n ancilla qubits are used to store either $|c_j\rangle$ or $|c_j^{-1} \bmod N\rangle$. This is potentially unnecessary; $c_j = \prod_{i=1}^{d} a_i^{z_{i,j}}$ is a product of at most d integers of $O(\log d)$ bits each, so c_j only comprises $O(d \log d) = \widetilde{O}(\sqrt{n})$ bits. While the same cannot be said of $c_j^{-1} \bmod N$, it might be possible to instead store $\prod_{i=1}^{d} a_i^{1-z_{i,j}}$ (which also only comprises $\widetilde{O}(\sqrt{n})$ bits) in its place, and bake the fixed multiplier $(\prod_{i=1}^{d} a_i^{-1}) \bmod N$ into the circuit architecture.

   We used n qubits for this register so that we could maintain a clean abstraction for multiplication operations, namely multiplying two n-bit integers that are reduced mod N. It may be possible to improve this if the multiplication algorithm being used can multiply a small integer with an n-bit integer without padding to make both integers n bits.

6 Error-Resilience for Regev's Classical Postprocessing

In this section, we prove Lemma 4.4, our main error correction lemma for Regev's postprocessing algorithm. We start by introducing our error detection and filtering algorithm. The intuition behind it is that uncorrupted samples will be very close to $\Lambda^*/\mathbb{Z}^d$, so it should be easy to find small integer relations between them. Moreover, it should be much harder to find a small integer relation between our uncorrupted samples and even one corrupted sample. We formalize this as a shortest-vector problem in a suitably constructed lattice, and will see that the approximate solution to this via LLL [LLL82] is sufficient for our purposes. Using this idea, we can identify sufficiently many uncorrupted samples and apply Regev's postprocessing procedure [Reg23] to these. Our algorithm is described in Algorithm 6.1.

Our analysis proceeds in three steps:

1. In Section 6.1, we show that with probability $1 - o(1)$, Algorithm 6.1 will add at least 1 sample to B in each iteration until $|B| \geq \gamma d$, hence proving that it terminates in polynomial time.

2. In Section 6.2, we formalize the above intuition and argue that none of the samples added to B will be corrupted, with probability $1 - o(1)$.

3. Although we use Regev's postprocessing procedure as-is, it turns out that Regev's analysis is not directly applicable to our algorithm. In Section 6.3, we outline the necessary modifications and elaborate in Appendix B.

Algorithm 6.1: Classical postprocessing of the outputs from Regev's quantum circuit

---

Data: Norm bound T and noisy samples $w_1, w_2, \ldots, w_m \in \mathbb{R}^d/\mathbb{Z}^d$ from $\Lambda^*/\mathbb{Z}^d$ corrupted according to the model in Section 4.2.1.

Result: A finite list of elements of Λ, such that they generate any element $u \in \Lambda$ with $||u||_2 \leq T$.

Initialize $B = \emptyset$. B will track the subset of samples that we select. Let $S = 2^{An/d}$. Recall that $m = \alpha d$.

1. Repeat the following until $|B| \geq \gamma d$:

   (a) Let E be a subset of $[m]$ such that $E \cap B = \emptyset$ and $|E| = (\alpha - \gamma)d$. This exists because $|B| \leq \gamma d$.

   (b) Define the matrix $W \in \mathbb{R}^{d \times |E|}$ to be the matrix with columns w_i for $i \in E$.

   (c) Define the lattice $\Lambda'' \subseteq \mathbb{R}^{d+|E|}$ as that spanned by the columns of

   $$H = \left(\begin{array}{c|c} SI_{d\times d} & SW \\ \hline 0 & I_{|E|\times|E|} \end{array}\right).$$

   We note that this is similar but not identical to the lattice construction used in Regev's postprocessing procedure [Reg23].

   (d) Apply LLL basis reduction [LLL82] to Λ'' to find a $2^{(d+|E|-1)/2}$-approximate shortest vector in Λ''.

   (e) This vector can be written as

   $$H \left(\begin{array}{c} \beta \\ \hline a_i : i \in E \end{array}\right) = \left(\begin{array}{c} S(\beta + \sum_{i\in E} a_i w_i) \\ \hline a_i : i \in E \end{array}\right),$$

   where $\beta \in \mathbb{Z}^d$ is some vector and $a_i : i \in E$ denotes a column vector in $\mathbb{Z}^{|E|}$.

   (f) For each $i \in E$, add i to B if and only if $a_i \neq 0$.

2. Let $B' \subseteq B$ be arbitrary with $|B'| = \gamma d$. Apply Regev's classical postprocessing (Algorithm B.1) to the collection $\{w_i : i \in B'\}$.

6.1 Proof that Our Algorithm Selects γd Samples

Firstly, note that since $p < 1 - (\gamma+1)/\alpha$, with probability $1 - o(1)$ at most $(\alpha - \gamma - 1)d$ of our samples from the quantum circuit will be corrupted. We now show an upper bound on the shortest vector of Λ'' in the next two lemmas:

Lemma 6.1. *Given vectors $v_1, \ldots, v_d \in \Lambda^*$, there exist $a_1, a_2, \ldots, a_d \in \mathbb{Z}$, not all zero, such that $|a_i| \leq 2^{n/d}$ for all i and $\sum_i a_i v_i \in \mathbb{Z}^d$.*

Proof. This follows from a pigeonhole principle argument similar to that observed by [Reg23], which we restate here. Consider the sums $\sum_{i=1}^d a_i v_i \in \Lambda^*$ for all $|a_i| \leq 2^{n/d-1}$. There are $(2^{n/d}+1)^d > 2^n > \det \Lambda = |\Lambda^*/\mathbb{Z}^d|$ such sums, so some two belong in the same coset of $\Lambda^*/\mathbb{Z}^d$. The difference between those sums will hence lie in $\mathbb{Z}^d$, and each coefficient will have magnitude $\leq 2^{n/d}$. At least one coefficient will be nonzero. □

Lemma 6.2. *With probability $1 - o(1)$, for any E with $|E| = (\alpha - \gamma)d$ there exists a nonzero vector in the corresponding Λ'' of length at most $2^{(1+o(1))n/d}$.*

Proof. Observe firstly that if $|E| = (\alpha - \gamma)d$ and we know that with probability $1 - o(1)$ we have at most $(\alpha - \gamma - 1)d$ corrupted samples in total, this implies that there are $\geq d$ uncorrupted samples in E. By Lemma 6.1, it follows that there exist integers a_i for $i \in E$ such that:

- $\sum_{i \in E} a_i v_i \in \mathbb{Z}^d$.
- $|a_i| \leq 2^{n/d}$ for all i.
- a_i is nonzero for at most d values of i, and these indices correspond to uncorrupted samples.

Hence we can set $\beta = -\sum_{i \in E} a_i v_i \in \mathbb{Z}^d$. Then consider the lattice element

$$H\left(\frac{\beta}{a_i : i \in E}\right) = \left(\frac{S(\beta + \sum_{i \in E} a_i w_i)}{a_i : i \in E}\right).$$

This is nonzero and an element of Λ''. Moreover, we have:

$$\beta + \sum_{i \in E} a_i w_i = \beta + \sum_{i \in E} a_i v_i + \sum_{i \in E} a_i \delta_i = \sum_{i \in E} a_i \delta_i,$$

which has norm at most $m 2^{n/d} \cdot 2^{(-A+o(1))n/d} = m 2^{(-A+1+o(1))n/d}$. (Note that a_i is 0 for any i corresponding to a corrupted sample). Finally, we have $\sum_{i \in E} a_i^2 \leq d \cdot 2^{2n/d}$. It follows that the norm of this element in Λ'' is at most:

$$\begin{aligned}\sqrt{S^2 m^2 2^{(-2A+2+o(1))n/d} + d \cdot 2^{2n/d}} &= \sqrt{m^2 2^{(2+o(1))n/d} + d \cdot 2^{2n/d}} \\ &\leq \sqrt{(m^2+d) 2^{(2+o(1))n/d}} \\ &= 2^{(1+o(1))n/d}.\end{aligned}$$

Lemma 6.3. *With probability $1 - o(1)$, as long as $|B| < \gamma d$, our algorithm will add at least one sample to B in each iteration.*

Proof. By Lemma 6.2, our algorithm will find a nonzero vector in Λ'' of ℓ_2 norm at most

$$2^{(d+|E|-1)/2} \cdot 2^{(1+o(1))n/d} \leq 2^{(\alpha-\gamma+3+o(1))n/(2d)}.$$

We can write this vector as

$$H\begin{pmatrix} \beta \\ a_i : i \in E \end{pmatrix} = \begin{pmatrix} S(\beta + \sum_{i \in E} a_i w_i) \\ a_i : i \in E \end{pmatrix}.$$

Suppose for the sake of contradiction that our algorithm adds no samples to B in this round. Then a_i must be 0 for all $i \in E$. It follows that $\beta + \sum_{i \in E} a_i w_i = \beta$ must be a nonzero vector in $\mathbb{Z}^d$. It follows that the norm of this vector is at least $S||\beta||_2 \geq S = 2^{An/d}$, which is a contradiction for $A \geq \frac{\alpha-\gamma+3}{2} + o(1)$. □

It follows that in each round of the algorithm, $|B|$ will increase by at least 1. This implies that Algorithm 6.1 will terminate in polynomial time, and it will find a subset of $\geq \gamma d$ samples to pass to Regev's postprocessing procedure [Reg23].

6.2 Proof that Our Selected Samples are Not Corrupted

Our next step is to show that our algorithm will not select any corrupted samples to be passed to Regev's postprocessing procedure. We address this in the next two lemmas.

Lemma 6.4. *With probability $1 - o(1)$, there do not exist integers $a_1, a_2, \ldots, a_m$ such that:*

- $|a_i| \leq 2^{(\alpha-\gamma+3+o(1))n/(2d)}$ *for all i.*
- $a_i \neq 0$ *for at least one index i corresponding to a corrupted sample.*
- $d(\sum_{i \text{ corrupted}} a_i \epsilon_i, \Lambda^*/\mathbb{Z}^d) \leq 2^{(-2A+\alpha-\gamma+3+o(1))n/(2d)}$.

Proof. This directly follows from the assumption that $\mathcal{D}$ (the distribution of each ϵ_i) is well-spread, after conditioning on the subset of our m samples that are corrupted. The assumption can be then applied with k set to the number of corrupted samples, which is clearly $\leq \alpha d$. □

Lemma 6.5. *With probability $1 - o(1)$, as long as $|B| < \gamma d$, our algorithm will not add any corrupted samples to B in any iteration.*

Proof. As in the proof of Lemma 6.3, it follows from Lemma 6.2 that our algorithm will find a nonzero vector in Λ'' of ℓ_2 norm at most

$$2^{(d+|E|-1)/2} \cdot 2^{(1+o(1))n/d} \leq 2^{(\alpha-\gamma+3+o(1))n/(2d)},$$

and moreover we can write this vector as

$$H\begin{pmatrix} \beta \\ a_i : i \in E \end{pmatrix} = \begin{pmatrix} S(\beta + \sum_{i \in E} a_i w_i) \\ a_i : i \in E \end{pmatrix}.$$

Both of the following must hence hold:

- $|a_i| \leq 2^{(\alpha-\gamma+3+o(1))n/(2d)}$ for all $i \in E$.
- $||\beta + \sum_{i\in E} a_i w_i||_2 \leq S^{-1}2^{(\alpha-\gamma+3+o(1))n/(2d)} = 2^{(-2A+\alpha-\gamma+3+o(1))n/(2d)}$.

Letting $\mathcal{C}$ denote the set of corrupted samples, we can write the latter of these two expressions as follows:

$$\begin{aligned}\beta + \sum_{i\in E} a_i w_i &= \beta + \sum_{i\in E\cap\overline{\mathcal{C}}} a_i(v_i+\delta_i) + \sum_{i\in E\cap\mathcal{C}} a_i(\eta(v_i+\delta_i)+\epsilon_i) \\ &= \left(\beta + \sum_{i\in E\cap\overline{\mathcal{C}}} a_i v_i + \sum_{i\in E\cap\mathcal{C}} \eta a_i v_i\right) + \left(\sum_{i\in E\cap\overline{\mathcal{C}}} a_i\delta_i + \sum_{i\in E\cap\mathcal{C}} \eta a_i\delta_i\right) + \left(\sum_{i\in E\cap\mathcal{C}} a_i\epsilon_i\right)\end{aligned}$$

Rearranging, we get

$$\begin{aligned}\sum_{i\in E\cap\mathcal{C}} a_i\epsilon_i = \left(\beta + \sum_{i\in E} a_i w_i\right) - \left(\sum_{i\in E\cap\overline{\mathcal{C}}} a_i\delta_i + \sum_{i\in E\cap\mathcal{C}} \eta a_i\delta_i\right) \\ - \left(\beta + \sum_{i\in E\cap\overline{\mathcal{C}}} a_i v_i + \sum_{i\in E\cap\mathcal{C}} \eta a_i v_i\right)\end{aligned}$$

The last of these three terms is some element in $\Lambda^*/\mathbb{Z}^d$, noting that η and the a_i's are all integers and that $\beta \in \mathbb{Z}^d$. We can bound the second term as follows:

$$\begin{aligned}\left\|\sum_{i\in E\cap\overline{\mathcal{C}}} a_i\delta_i + \sum_{i\in E\cap\mathcal{C}} \eta a_i\delta_i\right\|_2 &\leq \alpha d \cdot \max_i |a_i| \cdot \max_i ||\delta_i||_2 \\ &\leq \alpha d \cdot 2^{(\alpha-\gamma+3+o(1))n/(2d)} \cdot 2^{(-A+o(1))n/d} \\ &= 2^{(-2A+\alpha-\gamma+3+o(1))n/(2d)}.\end{aligned}$$

Finally, we already observed that the first term also has magnitude at most $2^{(-2A+\alpha-\gamma+3+o(1))n/(2d)}$. It follows by the triangle inequality that

$$d\left(\sum_{i\in E\cap\mathcal{C}} a_i\epsilon_i, \Lambda^*/\mathbb{Z}^d\right) \leq 2^{(-2A+\alpha-\gamma+3+o(1))n/(2d)}.$$

By Lemma 6.4, this forces $a_i = 0$ for all $i \in E$ such that sample i is corrupted. This implies that our algorithm will not add any corrupted samples to B. □

6.3 Proof that Regev's Postprocessing Works with Any γd Uncorrupted Samples

Regev's postprocessing argument [Reg23] works with $d+4$ independent and uncorrupted samples from $\Lambda^*/\mathbb{Z}^d$ and proceeds as follows:

1. First, he applies a result due to Pomerance [Pom02] to argue that these $d+4$ elements will generate $\Lambda^*/\mathbb{Z}^d$ with probability at least $1/2$.

2. He then argues that it is unlikely for there to exist $u \in \mathbb{Z}^d \setminus \Lambda$ which is nearly orthogonal to all $d+4$ samples. Thus these samples can be used to construct a suitable lattice such that applying LLL basis reduction [LLL82] yields a basis for short vectors in Λ.

In our case, we do not quite have independence; instead, what we have is a subset of size γd out of αd independent samples from $\Lambda^*/\mathbb{Z}^d$. Our analysis in Section 6.2 implies that we can assume these samples are uncorrupted. Informally, we just need to argue that this subset is "close enough" to independent that all of Regev's analysis still works. This largely mirrors the existing analysis in [Reg23], so we defer details to Appendix B.

Acknowledgements. We thank Oded Regev and Martin Ekerå for their detailed comments on this manuscript. SR would also like to thank Ittai Rubinstein for helpful discussions. SR was supported by an Akamai Presidential Fellowship. VV was supported in part by DARPA under Agreement Number HR00112020023, NSF CNS-2154149, a grant from the MIT-IBM Watson AI, a Thornton Family Faculty Research Innovation Fellowship from MIT and a Simons Investigator Award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

References

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, October 2019. 3

[Acc96] Vincenzo Acciaro. The probability of generating some common families of finite groups. *Utilitas Mathematica*, pages 243–254, 1996. 40, 41, 42

[BCDP96] David Beckman, Amalavoyal N Chari, Srikrishna Devabhaktuni, and John Preskill. Efficient networks for quantum factoring. *Physical Review A*, 54(2):1034, 1996. 2, 4

[Bea03] Stéphane Beauregard. Circuit for Shor's algorithm using 2n+3 qubits. *Quantum Inf. Comput.*, 3(2):175–185, 2003. 2, 4

[BMT+07] Andrew Byrne, Nicolas Meloni, Arnaud Tisserand, Emanuel M. Popovici, and William Peter Marnane. Comparison of simple power analysis attack resistant algorithms for an elliptic curve cryptosystem. *Journal of Computers*, 2(10), December 2007. 4, 16

[Cai23] Jin-Yi Cai. Shor's algorithm does not factor large integers in the presence of noise. *CoRR*, abs/2306.10072, 2023. 3

[CCHL22] Sitan Chen, Jordan Cotler, Hsin-Yuan Huang, and Jerry Li. The complexity of NISQ. *CoRR*, abs/2210.07234, 2022. 3

[CKM19] Earl Campbell, Ankur Khurana, and Ashley Montanaro. Applying quantum algorithms to constraint satisfaction problems. *Quantum*, 3:167, 2019. 12

[Cop02] Don Coppersmith. An approximate Fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067*, 2002. 2, 10, 11

[CW00] Richard Cleve and John Watrous. Fast parallel circuits for the quantum fourier transform. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 526–536. IEEE Computer Society, 2000. 2

[Dra00] Thomas G Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000. 21, 35

[EG24] Martin Ekerå and Joel Gärtner. Extending Regev's factoring algorithm to compute discrete logarithms, 2024. 7, 15, 45

[EH17] Martin Ekerå and Johan Håstad. Quantum algorithms for computing short discrete logarithms and factoring RSA integers. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017, Proceedings*, volume 10346 of *Lecture Notes in Computer Science*, pages 347–363. Springer, 2017. 2

[FMMC12] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012. 12

[GE21] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021. 2, 3, 12

[Gid17] Craig Gidney. Factoring with $n+2$ clean qubits and $n-1$ dirty qubits. *arXiv preprint arXiv:1706.07884*, 2017. 2, 4

[Gid19] Craig Gidney. Asymptotically efficient quantum Karatsuba multiplication. *arXiv preprint arXiv:1904.07356*, 2019. 2, 4, 5, 33, 34, 37

[Gid23] Craig Gidney. Comment on Scott Aaronson's blog, 2023. 2

[GR02] Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions, 2002. 9

[HRS17] Thomas Häner, Martin Roetteler, and Krysta M. Svore. Factoring using $2n+2$ qubits with Toffoli based modular multiplication. *Quantum Inf. Comput.*, 17(7&8):673–684, 2017. 2, 4, 16

[HvdH21] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2), March 2021. 2, 4, 5, 7, 25, 33, 34, 37

[Kal17a] Burton S. Kaliski Jr. A quantum "magic box" for the discrete logarithm problem. *Cryptology ePrint Archive*, 2017. 7

[Kal17b] Burton S. Kaliski Jr. Targeted Fibonacci exponentiation. *arXiv preprint arXiv:1711.02491*, 2017. 4, 6, 7, 15, 16, 19, 44, 45

[Kle08] Shmuel T. Klein. Should one always use repeated squaring for modular exponentiation? *Information Processing Letters*, 106(6):232–237, June 2008. 4, 16

[KO62] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145, pages 293–294. Russian Academy of Sciences, 1962. 5

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982. 25, 26, 30, 39

[Mel07] Nicolas Meloni. New point addition formulae for ECC applications. In *Arithmetic of Finite Fields: First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007. Proceedings 1*, pages 189–201. Springer, 2007. 4, 16

[Pom02] Carl Pomerance. The expected number of random elements to generate a finite abelian group. *Periodica Mathematica Hungarica*, 43:191–198, 2002. 29, 42

[PZ03] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Inf. Comput.*, 3(4):317–344, 2003. 7

[RC18] Rich Rines and Isaac Chuang. High performance quantum modular multipliers. *arXiv preprint arXiv:1801.01081*, 2018. 7, 16

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009. 9

[Reg23] Oded Regev. An efficient quantum factoring algorithm. *arXiv preprint arXiv:2308.06572*, 2023. 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 21, 25, 26, 27, 28, 29, 30, 38, 39, 40, 41, 45, 46

[RNSL17] Martin Roetteler, Michael Naehrig, Krysta M Svore, and Kristin Lauter. 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, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pages 241–270. Springer, 2017. 4, 5, 36, 37

[Sei01] Jean-Pierre Seifert. Using fewer qubits in Shor's factorization algorithm via simultaneous Diophantine approximation. In *Cryptographers' Track at the RSA Conference*, pages 319–327. Springer, 2001. 2

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994. 3

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. 1, 4, 7, 16, 25

[SS71] Arnold Schönhage and Volker Strassen. Fast multiplication of large numbers. *Computing*, 7:281–292, 1971. 7

[TK06] Yasuhiro Takahashi and Noboru Kunihiro. A quantum circuit for Shor's factoring algorithm using 2n+ 2 qubits. *Quantum Information & Computation*, 6(2):184–192, 2006. 2, 4

[VBE96] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1):147, 1996. 2, 4

[Zec72] Édouard Zeckendorf. Representations of natural numbers by a sum of Fibonacci numbers and Lucas numbers. *Bulletin of the Royal Society of Sciences of Liege*, pages 179–182, 1972. 48

A Implementation of Our Multiplication Oracle

In Theorem 1.1 and our algorithm, we assume a quantum multiplication circuit that takes a specific form, mapping

$$|a\rangle\,|b\rangle\,|t\rangle\,|0^S\rangle \mapsto |a\rangle\,|b\rangle\,|(t+ab) \bmod N\rangle\,|0^S\rangle \ .$$

Existing multiplication circuits such as those by [HvdH21] and [Gid19] may differ from this form in three ways:

- Circuits such as that by [HvdH21] may be classical circuits rather than quantum.
- These circuits, as in both [HvdH21] and [Gid19], may only support multiplication over $\mathbb{Z}$ rather than modulo N i.e. they produce ab as an output rather than $ab \bmod N$.
- Even if these circuits could compute $ab \bmod N$, they may not have the specific structure mapping $|t\rangle$ to $|(t + ab) \bmod N\rangle$ that we require.

In this section, we show how to overcome each of these gaps:

- In Section A.1, we show how to "compile" a classical circuit into a quantum circuit that can compute the same function out of place.
- In Section A.2, we show how to go from computing ab to computing $ab \bmod N$.
- In Section A.3, we show how to achieve the specific structure computing $(t + ab) \bmod N$ that we need.

Additionally, we quickly verify in Section A.4 that these arguments yield the space and size guarantees we claim when using the multiplication circuits in [HvdH21] and [Gid19]. Finally, we present a different oracle construction for the special case of schoolbook multiplication in Section A.5. This avoids the space overheads associated with the above "conversion" procedures, and only requires $O(1)$ ancilla qubits.

A.1 Compiling Classical Circuits into Quantum Circuits

In this section, we state and reprove the well-known fact that a classical circuit can be "compiled" into a quantum circuit that can carry out the same computation in superposition, while tracking the space and size of such a quantum circuit.

Lemma A.1. *(Well-known) Assume there exists a classical circuit with G wires (including input and output bits) that computes some function $f(x) \in \{0,1\}^{n_o}$ of its input $x \in \{0,1\}^{n_i}$. Then there exists a quantum circuit using $O(G)$ gates mapping*

$$|x\rangle |0^{G+n_o-n_i}\rangle \mapsto |x\rangle |0^{G-n_i}\rangle |f(x)\rangle .$$

Note that the number of qubits in such a circuit is $G + n_o \leq 2G$ (since G includes output wires).

Proof. Assume without loss of generality that the classical circuit comprises only NOT and AND gates.

Let us temporarily ignore n_o of the ancilla qubits. The remaining ancilla qubits can be assigned bijectively to non-input wires in the classical circuit. Using these qubits, we simulate the classical circuit as follows:

- For a NOT gate with input wire corresponding to qubit $|a\rangle$ and output wire corresponding to qubit $|b\rangle$, we can apply a CNOT gate with control qubit $|a\rangle$ and target qubit $|b\rangle$, and then apply an X gate to $|b\rangle$.

- For an AND gate with input wires corresponding to qubits $|a\rangle, |b\rangle$ and output wire corresponding to qubit $|c\rangle$, we can apply a CCNOT gate with control qubits $|a\rangle, |b\rangle$ and target qubit $|c\rangle$.

At the end of this computation, we will have the state

$$|x\rangle |\phi\rangle |f(x)\rangle |0^{n_o}\rangle,$$

for some state ϕ on $G - n_i - n_o$ qubits. We can then copy $|f(x)\rangle$ to the n_o remaining ancilla qubits using CNOT gates, and then uncompute the classical circuit simulation to obtain the final desired state. The number of gates in our quantum circuit is clearly $O(G)$. □

A.2 Computing $ab \bmod N$

To go from computing ab to computing $ab \bmod N$, the main ingredient is unsurprisingly an efficient quantum circuit for out-of-place reduction modulo N:

Lemma A.2. *Assume there exists a quantum circuit mapping*

$$|a\rangle |b\rangle |0^{2n}\rangle |0^S\rangle \mapsto |a\rangle |b\rangle |ab\rangle |0^S\rangle$$

with G gates, where a, b are n-bit integers.

Let c be a $2n$-bit integer. Then there exists a quantum circuit using $O(G + n)$ gates mapping

$$|c\rangle |0^{S+O(n)}\rangle \mapsto |c\rangle |c \bmod N\rangle |0^{S+O(n)}\rangle.$$

Proof. Let $b = 2n$ be a precision parameter (so $c < 2^b$). Assume we have classically precomputed a high-precision approximation of $1/N$; concretely, we have an integer r so that

$$r/2^b < 1/N < (r+1)/2^b.$$

The idea is to use this to approximately divide c by N, which we do as follows:

$$\begin{aligned}
|c\rangle |r\rangle |N\rangle |0^{S+O(n)}\rangle &\to |c\rangle |r\rangle |N\rangle |cr\rangle |0^{S+O(n)}\rangle \\
&= |c\rangle |r\rangle |N\rangle |\lfloor cr/2^b \rfloor\rangle |cr \bmod 2^b\rangle |0^{S+O(n)}\rangle \\
&\quad \text{(splitting the register containing } cr \text{ into two pieces)} \\
&\to |c\rangle |r\rangle |N\rangle |\lfloor cr/2^b \rfloor\rangle |0^{S+O(n)}\rangle |cr \bmod 2^b\rangle |N \cdot \lfloor cr/2^b \rfloor\rangle |0^{S+O(n)}\rangle \\
&\to |c\rangle |r\rangle |N\rangle |\lfloor cr/2^b \rfloor\rangle |0^{S+O(n)}\rangle |cr \bmod 2^b\rangle |c - N \cdot \lfloor cr/2^b \rfloor\rangle |0^{S+O(n)}\rangle,
\end{aligned}$$

where the last step is achieved by using the inverse of an in-place integer addition circuit e.g. see [Dra00]. Next, observe that:

$$\begin{aligned}
N \cdot \lfloor cr/2^b \rfloor &\leq Ncr/2^b \\
&< c \\
\Rightarrow 0 &< c - N \cdot \lfloor cr/2^b \rfloor, \text{ and}
\end{aligned}$$

$$\begin{aligned} N \cdot \lfloor cr/2^b \rfloor &> N \cdot (cr/2^b - 1) \\ &> N \cdot (c \cdot (1/N - 1/2^b) - 1) \\ &> c - 2N \\ \Rightarrow 2N &< c - N \cdot \lfloor cr/2^b \rfloor. \end{aligned}$$

So the number we have computed is congruent to c modulo N, and is contained in the interval $(0, 2N)$. This implies that it is either equal to $c \bmod N$ or $(c \bmod N) + N$. These cases can be distinguished and $c \bmod N$ computed out of place using $O(n)$ qubits and $O(n)$ gates. Finally, this can be copied to a fresh register and the above computation uncomputed.

In terms of circuit size, the only operation that is not $O(n)$ gates is multiplying pairs of integers of $\leq 2n$ bits a constant number of times. This can in turn be done using a constant number of calls to the given quantum circuit (one call does not suffice, since we are assuming a circuit that multiplies n-bit integers rather than $2n$-bit). The conclusion follows. □

Now, it is straightforward to check that we can compute $ab \bmod N$:

Lemma A.3. *Using the same multiplication circuit as in Lemma A.2, there exists a quantum circuit using $O(G + n)$ gates mapping*

$$|a\rangle |b\rangle |0^n\rangle |0^{S+O(n)}\rangle \mapsto |a\rangle |b\rangle |ab \bmod N\rangle |0^{S+O(n)}\rangle .$$

Proof. We proceed as follows:

$$\begin{aligned} |a\rangle |b\rangle |0^n\rangle |0^{S+O(n)}\rangle &\mapsto |a\rangle |b\rangle |ab\rangle |0^n\rangle |0^{S+O(n)}\rangle \text{ (using the multiplication circuit)} \\ &\mapsto |a\rangle |b\rangle |ab\rangle |ab \bmod N\rangle |0^{S+O(n)}\rangle , \text{ (using Lemma A.2)} \end{aligned}$$

after which we can copy and uncompute as usual to obtain the desired result. □

A.3 Computing $(t + ab) \bmod N$

Now, we can assume we have a quantum circuit that maps

$$|a\rangle |b\rangle |0^n\rangle |0^{S_0}\rangle \mapsto |a\rangle |b\rangle |ab \bmod N\rangle |0^{S_0}\rangle .$$

We now show how to obtain our specific functionality from this, at the expense of n additional ancilla qubits. The key ingredients for our constructions in this section are the following space-efficient primitives for modular doubling and addition, due to [RNSL17]:

Lemma A.4. [RNSL17] *There exists a circuit using $O(n \log n)$ gates computing the mapping*

$$|x\rangle |y\rangle |0\rangle |0\rangle \mapsto |x\rangle |(x + y) \bmod N\rangle |0\rangle |0\rangle .$$

Here, x and y are reduced mod N, and each $|0\rangle$ is just one individual ancilla qubit.

Lemma A.5. [RNSL17] *Provided N is odd, there exists a circuit using $O(n \log n)$ gates computing the mapping*

$$|x\rangle |0\rangle |0\rangle \mapsto |2x \bmod N\rangle |0\rangle |0\rangle .$$

Here, x is once again reduced mod N, and we only have two ancilla qubits.

Using the addition lemma, we have our first simple result:

Lemma A.6. *Suppose we have a quantum circuit using G_0 gates that maps*

$$|a\rangle\,|b\rangle\,|0^n\rangle\,|0^{S_0}\rangle \mapsto |a\rangle\,|b\rangle\,|ab \bmod N\rangle\,|0^{S_0}\rangle\,,$$

and $S_0 \geq 2$. Then there is also a quantum circuit using $O(G_0 + n \log n)$ gates that maps

$$|a\rangle\,|b\rangle\,|t\rangle\,|0^{S_0+n}\rangle \mapsto |a\rangle\,|b\rangle\,|(t+ab) \bmod N\rangle\,|0^{S_0+n}\rangle\,,$$

whenever t is reduced mod N.

Proof. We proceed as follows:

$$\begin{aligned}
&|a\rangle\,|b\rangle\,|t\rangle\,|0^n\rangle\,|0^{S_0}\rangle \\
&\quad\to |a\rangle\,|b\rangle\,|t\rangle\,|ab \bmod N\rangle\,|0^{S_0}\rangle \text{ (using the given circuit)} \\
&\quad\to |a\rangle\,|b\rangle\,|(t+ab) \bmod N\rangle\,|ab \bmod N\rangle\,|0^{S_0}\rangle \text{ (using Lemma A.4)} \\
&\quad\to |a\rangle\,|b\rangle\,|(t+ab) \bmod N\rangle\,|0^n\rangle\,|0^{S_0}\rangle \text{ (using the inverse of the given circuit)}
\end{aligned}$$

The number of gates is clearly $O(G_0 + n \log n)$, so this establishes the lemma. □

A.4 Special Cases of [HvdH21] and [Gid19]

Let us check that the compilation procedures presented here yield the space and size guarantees we claimed in Theorem 1.1 when adapting the multiplication circuits by [HvdH21] and [Gid19] to our situation.

In the case of [HvdH21], we initially have a classical circuit of size $O(n \log n)$. Section A.1 converts this into a quantum circuit with size and space $O(n \log n)$ that computes ab. As for [Gid19], we initially have a quantum circuit with size $O(n^{\log_2 3})$ and space $O(n)$ for computing ab.

Now applying the conversions in Section A.2 and Section A.3 yields circuits computing our desired $t \mapsto (t+ab) \bmod N$ functionality while retaining the asymptotic size and space guarantees; the size incurs a constant factor overhead plus $O(n \log n)$ extra gates, while the space incurs an $O(n)$ additive overhead.

A.5 Special Case of Schoolbook Multiplication

Finally, we show how to construct a circuit using schoolbook multiplication where only $O(1)$ ancilla qubits are needed. [RNSL17] already constructs a circuit mapping $|a\rangle\,|b\rangle\,|0^n\rangle \mapsto |a\rangle\,|b\rangle\,|ab \bmod N\rangle$, but we want to avoid the $O(n)$-qubit overhead of the conversion procedure in Section A.3. Fortunately, this can be done with just a slight tweak of their shift-and-add construction:

Lemma A.7. *Provided N is odd, there exists a quantum circuit using $O(n^2 \log n)$ gates that maps*

$$|a\rangle\,|b\rangle\,|t\rangle\,|0\rangle\,|0\rangle \mapsto |a\rangle\,|b\rangle\,|(t+ab) \bmod N\rangle\,|0\rangle\,|0\rangle\,,$$

whenever a, b, t are all reduced mod N.

Proof. Label the bits comprising a as $a_0, a_1, \ldots, a_{n-1}$, so that $a = \sum_{i=0}^{n-1} a_i 2^i$. Then observe that

$$ab = \left(\sum_{i=0}^{n-1} a_i 2^i\right) b = \sum_{i=0}^{n-1} a_i \cdot (2^i b).$$

This suggests the following algorithm:

1. Let x, y, z denote the values in the register containing a, b, t respectively. So initially we have $x = a, y = b, z = t$.
2. Repeat the following for $i = 0, 1, \ldots, n-1$:
   (a) Use a_i as a control bit for the circuit in Lemma A.4 to update $z \leftarrow (z + a_i y) \bmod N$. We have the necessary two ancilla qubits for this.
   (b) Use Lemma A.5 to update $y \leftarrow 2y \bmod N$. We have the necessary two ancilla qubits for this.
3. Repeat the following n times: update $y \leftarrow y/2 \bmod N$. This can be done using the inverse of the circuit from Lemma A.5. Once again, this only needs two ancilla qubits.

First, note that we make $O(n)$ calls to the circuits in Lemmas A.4 and A.5, implying the gate complexity of $O(n^2 \log n)$.

To see correctness, it is straightforward to show by induction that at the end of step i within the loop in stage 2, we will have:

$$\begin{aligned} &x = a, \\ &y = 2^{i+1} b \bmod N, \text{ and} \\ &z = \left(t + \sum_{j=0}^{i} a_j \cdot 2^j b\right) \bmod N. \end{aligned}$$

So at the end of stage 2, we will have $x = a$, $y = 2^n b \bmod N$, and $z = (t + ab) \bmod N$. Then after stage 3, y will become b again. The ancilla qubits are returned to 0 in each step, so the final state after applying our algorithm will be

$$|x\rangle |y\rangle |z\rangle |0\rangle |0\rangle = |a\rangle |b\rangle |(t + ab) \bmod N\rangle |0\rangle |0\rangle ,$$

as desired. □

B Analysis of Regev's Classical Postprocessing Procedure

In this section, we state Regev's classical postprocessing procedure and adapt the analysis by [Reg23] to work for our slightly different setting. As explained in Section 6.3, Regev's analysis assumes his postprocessing procedure is given $k = d + 4$ independent noisy samples from $\Lambda^*/\mathbb{Z}^d$. Our setting is different because we have an arbitrary subset of size $k = \gamma d$ out of $m = \alpha d$ independent noisy

Algorithm B.1: Regev's classical postprocessing procedure [Reg23]

Data: Norm bound T and a collection of noisy samples $w_1, w_2, \ldots, w_k \in \mathbb{R}^d/\mathbb{Z}^d$ from $\Lambda^*/\mathbb{Z}^d$.

Result: A finite list of elements of Λ, such that they generate any element $u \in \Lambda$ with $||u||_2 \leq T$.

1. Define the lattice $\Lambda' \subseteq \mathbb{R}^{d+k}$ as that spanned by the columns of

$$\left(\begin{array}{c|c} I_{d\times d} & 0 \\ \hline \delta^{-1}w_1 & \\ \vdots & I_{k\times k} \\ \delta^{-1}w_k & \end{array}\right).$$

   Recall that δ is an upper bound on the magnitude of additive noise δ_i included in each w_i.

2. Let $z_1, \ldots, z_{d+k} \in \mathbb{R}^{d+k}$ be an LLL reduced basis [LLL82] of Λ', and let $\tilde{z}_1, \ldots, \tilde{z}_{d+k}$ be the Gram-Schmidt orthogonalization of this basis.

3. Let $l \geq 0$ be minimal such that $||\tilde{z}_{l+1}||_2 \geq 2^{(d+k)/2} \cdot (k+1)^{1/2} \cdot T$ (if no such l exists, take $l = d + k$).

4. For each $i \in [d+k]$, let $z'_i \in \mathbb{R}^d$ consist of the first d coordinates of z_i. Output $z'_1, \ldots, z'_l$.

samples from $\Lambda^*/\mathbb{Z}^d$. We first describe Regev's postprocessing procedure [Reg23] in Algorithm B.1, then turn to its analysis.

To analyze this algorithm in the way it is used in Algorithm 6.1, we need to prove the following lemma:

Lemma B.1. *(Compare with Lemma 3.4, which also analyzes Algorithm B.1) Let $\Lambda \subseteq \mathbb{Z}^d$, and let $T > 0$ be some norm bound. Assume we are given an arbitrary subset of $k = \gamma d$ out of $m = \alpha d$ independent samples of the form*

$$w_i = v_i + \delta_i,$$

where each v_i is a uniform sample from $\Lambda^*/\mathbb{Z}^d$ and δ_i is some additive error of magnitude at most δ. Additionally, assume the following inequalities:

- $\left(\frac{e\alpha}{\alpha-\gamma}\right)^{\alpha-\gamma} \cdot 2^{-\gamma+1} < 1.$
- $(\gamma d + d)^{1/2} \cdot 2^{(\gamma+1)d/2} \cdot (\gamma d + 1)^{1/2} \cdot T < \delta^{-1} \cdot (4 \det \Lambda)^{-1/(\gamma d)} \cdot 2^{-\alpha/\gamma}/6.$

Then, with probability at least $1/4$, Algorithm B.1 succeeds when given the subset of k samples as input i.e. it produces vectors $z_1, \ldots, z_l \in \Lambda$ that generate any element $u \in \Lambda$ with magnitude at most T.

We first set up some intermediate lemmas for the analysis necessary to prove Lemma B.1, closely mirroring the analysis by [Reg23]. The only nontrivial step in our adaptation is checking that this subset will generate $\Lambda^*/\mathbb{Z}^d$ with $\Omega(1)$ probability, which follows from the following lemma:

Lemma B.2. *(Compare with Corollary 4.2 in [Reg23]) Let G be an abelian group with a generating set (not necessarily minimal) of size d. Suppose we have $m = \alpha d$ independent and uniform samples from G, then with probability at least $1/2$, any γd of these samples will generate G.*

Proof. This follows from results by Acciaro [Acc96] on the probability that a certain number of samples will generate G. We defer details to Appendix C. □

The remainder of our adaptation of Regev's analysis [Reg23] is straightforwrad, but we detail it below for completeness.

Lemma B.3. *(Compare with Lemma 4.3 in [Reg23]) Assume $v_1, \dots, v_m$ are independent and uniform samples from $\Lambda^*/\mathbb{Z}^d$. Define a set $B' \subseteq [m]$ to be* good *if $|B'| = \gamma d$.*

Then with probability at least $1/4$, for any nonzero $u \in \mathbb{Z}^d/\Lambda$ and any good B', there exists an $i \in B'$ such that $\langle u, v_i \rangle \neq [-\epsilon, \epsilon] \bmod 1$, where $\epsilon = (4 \det \Lambda)^{-1/(\gamma d)} \cdot 2^{-\alpha/\gamma}/3$.

Proof. We almost exactly follow the proof by Regev [Reg23], except we also need to take a union bound over sets B'. We say B' is *generating* if the samples $\{v_i : i \in B'\}$ generate $\Lambda^*/\mathbb{Z}^d$.

We need to upper bound the probability of there existing nonzero $u \in \mathbb{Z}^d/\Lambda$ and good B' such that $\langle u, v_i \rangle \in [-\epsilon, \epsilon] \bmod 1$ for all $i \in B'$. By a union bound, this is at most the sum of the probabilities of the following two events:

- E_1: There exists a good set B' that is not generating.
- E_2: There exist nonzero $u \in \mathbb{Z}^d/\Lambda$ and B' that is both good and generating such that $\langle u, v_i \rangle \in [-\epsilon, \epsilon] \bmod 1$ for all $i \in B'$.

The abelian group $\Lambda^*/\mathbb{Z}^d$ has a generating set of size d e.g. by taking a basis of Λ^*. Lemma B.2 hence tells us that $\Pr[E_1] \leq 1/2$. So it remains to show that $\Pr[E_2] \leq 1/4$.

For E_2, temporarily fix a nonzero $u \in \mathbb{Z}^d/\Lambda$ and a good and generating subset $B' \subseteq [m]$. As argued by [Reg23], because u is not the zero coset, when we sample uniform v from $\Lambda^*/\mathbb{Z}^d$, $\langle u, v \rangle \bmod 1$ must be a uniform sample from $\{0, 1/t, \dots, (t-1)/t\}$ for some $t \geq 2$. There are two cases: if $t < 1/\epsilon$, then since B' is generating there must exist $i \in B'$ such that $\langle u, v_i \rangle \bmod 1 \neq 0 \Rightarrow \langle u, v_i \rangle \notin [-\epsilon, \epsilon] \bmod 1$, contradicting the hypothesis of E_2. If $t \geq 1/\epsilon$, then the probability of $\langle u, v \rangle \in [-\epsilon, \epsilon] \bmod 1$ will be $(1 + 2\lfloor t\epsilon \rfloor)/t \leq 3\epsilon$. Since the samples are independent, the probability that $\langle u, v_i \rangle \in [-\epsilon, \epsilon] \bmod 1$ for all $i \in B'$ is at most $(3\epsilon)^{|B'|} = (3\epsilon)^{\gamma d}$.

To finish, we take a union bound. There are trivially at most 2^m choices of the set B', and the number of choices of u is at most $|\mathbb{Z}^d/\Lambda| = \det \Lambda$. Hence we have:

$$\begin{aligned} \Pr[E_2] &\leq \det \Lambda \cdot 2^m \cdot (3\epsilon)^{\gamma d} \\ &= 1/4, \end{aligned}$$

which completes the proof of the lemma. □

Lemma B.4. *(Compare with Lemma 4.4 in [Reg23]) Assume Algorithm B.1 is given input as specified by Lemma 4.4. Then both of the following are true:*

- *For any $u \in \Lambda$, there exists $u' \in \Lambda'$ whose first d coordinates are equal to u and whose norm is at most $||u||_2 \cdot (k+1)^{1/2} = ||u||_2 \cdot (\gamma d + 1)^{1/2}$.*
- *With probability at least $1/4$ (over the randomness of $w_1, \ldots, w_m$), any nonzero $u' \in \Lambda'$ of norm $< \delta^{-1}\epsilon/2$ satisfies that its first d coordinates are a nonzero vector in Λ, where $\epsilon = (4 \det \Lambda)^{-1/(\gamma d)} \cdot 2^{-\alpha/\gamma}/3$.*

Proof. Identical to the proof by [Reg23]; we can just plug in the result from Lemma B.3 instead of Lemma 4.3 in [Reg23]. □

Lemma B.5. *(Follows directly from Claim 5.1 in [Reg23]) In Algorithm B.1, we have for all $i \in [l]$ that $||z_i||_2 \leq (d+k)^{1/2} \cdot 2^{(d+k)/2} \cdot (k+1)^{1/2} \cdot T$. Moreover, any vector in Λ' of norm at most $(k+1)^{1/2} \cdot T$ is expressible as an integer linear combination of $z_1, \ldots, z_l$.*

Proof. See [Reg23]. □

We can now put everything together, following the argument at the end of [Reg23]:

Proof of Lemma B.1. Consider any $u \in \Lambda$ such that $||u||_2 \leq T$. Then by Lemma B.4, there exists $u' \in \Lambda'$ whose first d coordinates are equal to u and with norm at most $(k+1)^{1/2} \cdot ||u||_2 \leq (k+1)^{1/2} \cdot T$. It follows by Lemma B.5 that there exist integers $\alpha_1, \ldots, \alpha_l$ such that:

$$\begin{aligned} u' &= \alpha_1 z_1 + \ldots + \alpha_l z_l \\ \Rightarrow u &= \alpha_1 z_1' + \ldots + \alpha_l z_l', \end{aligned}$$

by equating the first d coordinates. It hence remains to show that $z_i' \in \Lambda$ for all $i \in [l]$. To do this, note that Lemma B.5 also tells us for any $i \in [l]$ that:

$$\begin{aligned} ||z_i||_2 &\leq (d+k)^{1/2} \cdot 2^{(d+k)/2} \cdot (k+1)^{1/2} \cdot T \\ &= (\gamma d + d)^{1/2} \cdot 2^{(\gamma+1)d/2} \cdot (\gamma d + 1)^{1/2} \cdot T \\ &< \delta^{-1} \cdot (4 \det \Lambda)^{-1/(\gamma d)} \cdot 2^{-\alpha/\gamma}/6 \\ &= \delta^{-1}\epsilon/2. \end{aligned}$$

By Lemma B.4, it follows that with probability at least $1/4$, each z_i' must indeed be in Λ. □

C Proof of Lemma B.2

The probability that t independent and uniform samples from an abelian group G generate the group can be calculated exactly from results by Acciaro [Acc96]. We do this in the following two lemmas. For an abelian group G and integer r, let $\alpha_r(G)$ denote the probability that r uniform and independent samples from G generate G.

Lemma C.1. *(Lemma 4 in [Acc96], also stated by [Pom02]) Let G be a finite abelian group with order equal to some power of a prime p, and let r be the minimal number of generators for G. Then for any $t \geq r$, we have*

$$\alpha_t(G) = \prod_{i=t-r+1}^{t} (1 - p^{-i}).$$

Proof. This is not exactly the form that Acciaro's result is stated in, but Pomerance [Pom02] restates the result in this form in equation 2 of his paper. □

Lemma C.2. *Let G be a finite abelian group, and suppose there exists a generating set of size r for G (there may also be a strictly smaller generating set). Then for any $t \geq r$, we have*

$$\alpha_t(G) \geq \prod_{p||G|} (1 - p^{r-1-t})^r.$$

Proof. First, note that by writing G as a direct product of cyclic abelian groups of prime power order and then combining groups whose orders are powers of the same prime, we can write

$$G = G_{p_1} \times G_{p_2} \times \ldots \times G_{p_k},$$

where $p_1, \ldots, p_k$ are the distinct primes dividing $|G|$. Each G_{p_i} may not be cyclic, but its order will be a power of p_i. By Corollary 4 in [Acc96] and noting that the orders of the G_{p_i}'s are pairwise coprime, we have

$$\alpha_t(G) = \prod_{p||G|} \alpha_t(G_p).$$

It therefore suffices to show for each p that $\alpha_t(G_p) \geq (1 - p^{r-1-t})^r$. To do this, fix a p and let s be the minimal number of generators for G_p. We have $s \leq r$ (to see this, note that any r generators for G project down into a generating set of r elements for G_p, so a minimal generating set for G_p has size at most r). Now we can use Lemma C.1 to proceed as follows:

$$\begin{aligned}
\alpha_t(G_p) &= \prod_{i=t-s+1}^{t} (1 - p^{-i}) \\
&\geq \prod_{i=t-s+1}^{t} (1 - p^{r-1-t}) \text{ (since } i \geq t-s+1 \geq t-r+1) \\
&= (1 - p^{r-1-t})^s \\
&\geq (1 - p^{r-1-t})^r.
\end{aligned}$$

□

We can now complete the proof of Lemma B.2. Let $k = \gamma d$. We will bound $\alpha_k(G)$ by applying Lemma C.2 with $t = k$ and $r = d$. We hence have:

$$\alpha_k(G) \geq \prod_{p||G|} (1 - p^{d-1-k})^d$$

$$\geq \prod_{\text{all primes } p} (1 - p^{d-1-k})^d$$
$$= \frac{1}{\zeta(k+1-d)^d},$$

where we have used the Euler product formula for the Riemann zeta function ζ. Since $k+1-d$ is a real number > 1, we have:

$$\begin{aligned}
\zeta(k+1-d) &= \sum_{x=1}^{\infty} \frac{1}{x^{k+1-d}} \\
&\leq 1 + \frac{1}{2^{k+1-d}} + \int_2^{\infty} \frac{1}{x^{k+1-d}} dx \\
&= 1 + \frac{1}{2^{k+1-d}} + \left[\frac{x^{-k+d}}{k-d}\right]_{\infty}^{2} \\
&= 1 + 2^{-k+d-1} + \frac{2^{-k+d}}{k-d} \\
&\leq 1 + 2^{-k+d} \\
\Rightarrow \frac{1}{\zeta(k+1-d)} &\geq 1 - 2^{-k+d} \\
\Rightarrow \frac{1}{\zeta(k+1-d)^d} &\geq (1 - 2^{-k+d})^d \\
&\geq 1 - d \cdot 2^{-k+d} \text{ (by Bernoulli's inequality).}
\end{aligned}$$

Therefore, the probability that k uniform and independent samples from G do not generate G is

$$1 - \alpha_k(G) \leq d \cdot 2^{-k+d}.$$

Now to finish, let us take a union bound. The probability that out of $m = \alpha d$ uniform and independent samples, there exists a subset of size $k = \gamma d$ that does not generate G is at most:

$$\begin{aligned}
\binom{m}{k} \cdot d \cdot 2^{-k+d} &= \binom{m}{m-k} \cdot d \cdot 2^{-k+d} \\
&\leq \left(\frac{em}{m-k}\right)^{m-k} \cdot d \cdot 2^{-k+d} \\
&= \left(\frac{e\alpha}{\alpha-\gamma}\right)^{(\alpha-\gamma)d} \cdot d \cdot 2^{(-\gamma+1)d}
\end{aligned}$$

which is $< 1/2$ for d sufficiently large, provided that

$$\left(\frac{e\alpha}{\alpha-\gamma}\right)^{\alpha-\gamma} \cdot 2^{-\gamma+1} < 1.$$

This completes the proof of Lemma B.2. □

D Efficient Modular Fibonacci Exponentiation

In Section 5, we adapted ideas by Kaliski [Kal17b] to show how Fibonacci exponentiation could be used to efficiently compute a product of exponents on a quantum computer. It then follows that the same technique would also be helpful to compute a single exponent. We capture this in the following standalone result, and then discuss its applicability to other quantum algorithms.

As in Theorem 1.1, we assume throughout that we have a multiplication circuit that maps

$$|a\rangle |b\rangle |t\rangle |0^S\rangle \mapsto |a\rangle |b\rangle |(t + ab) \bmod N\rangle |0^S\rangle$$

where a, b, t are all n-bit integers and $0 \leq a, b, t < N$.

Lemma D.1. *There exists a quantum circuit using $O(mG + m^2)$ gates such that, for all n-bit integers a coprime to and reduced modulo N and for all m-bit integers z, it will map*

$$|a\rangle |a^{-1} \bmod N\rangle |z\rangle |0^M\rangle \mapsto |a\rangle |a^{-1} \bmod N\rangle |z\rangle |a^z \bmod N\rangle |0^{M-n}\rangle ,$$

where $M = S + O(m + n)$ is the number of ancilla qubits.

Proof. Our algorithm proceeds very similarly to Algorithm 5.2, so we just outline the key steps below and point out the space and size costs:

1. Construct $z_j \in \{0, 1\}$ for $j \leq \frac{m}{\log \phi}$ such that $z = \sum_j z_j F_j$. This can be done using the greedy algorithm in Lemma 5.5. This uses $O(m)$ ancilla qubits and comprises $O(m)$ additions/subtractions/comparisons on m-bit integers, for a total of $O(m^2)$ gates. The expression we want to compute is now

$$\prod_j a^{z_j F_j} \bmod N = \prod_j c_j^{F_j} \bmod N,$$

   where $c_j = a^{z_j} \in \{1, a\}$.

2. Follow Algorithm 5.1 to now compute the desired product. This uses $S + 5n = S + O(n)$ ancilla qubits and $O(mG)$ gates. Note importantly that since all the c_j's are either 1 or a, we only need $|\psi(a)\rangle = |a\rangle |a^{-1} \bmod N\rangle$, which we are given as part of the input.

3. Copy the final output $|a^z \bmod N\rangle$ to an n-bit register, and uncompute all previous operations to restore the original inputs. This requires another $O(n)$ ancilla qubits and essentially doubles the gate complexity (there is another $O(n)$ gates for the copying, but this will be dominated by G).

Hence the overall ancilla cost is $S + O(m + n)$ and the circuit size is $O(mG + m^2)$, as desired. □

We make some observations about our result below:

1. Exponentiation via the square-and-multiply algorithm involves $O(m)$ multiplications and hence requires $O(mG)$ gates. However, all intermediate results will have to be written out in separate registers, hence increasing the space demand to $S + O(mn)$ which is strictly worse than that of our construction.

We note that our result is not strictly better than square-and-multiply in the case that $m \gg G$; in this case, the $O(m^2)$ overhead from computing the Zeckendorf representation of the exponent will dominate the cost of the multiplications. This is not of concern when $m = O(n)$ or $m = O(\sqrt{n})$, as is the case in Shor's and Regev's algorithms respectively.

2. Precomputation-based approaches will be more efficient in situations where the set of possible values of a is small (or generated by a small subset of $\mathbb{Z}_N^*$). This is what happens in Shor's algorithm.

3. The need for $a^{-1} \bmod N$ to be available is potentially inconvenient. Kaliski's algorithm [Kal17b] does not require this, but it appears necessary when adapting to the quantum setting. Ideally, we would have a case like our implementation of Regev's algorithm where it is relatively straightforward to compute the inverses of all the bases we are interested in. However, in the worst case, one could use the extended Euclidean algorithm to compute $|a^{-1} \bmod N\rangle$, which may or may not be an acceptable overhead depending on the application.

E Applying Our Error Detection Procedure to Calculate Discrete Logarithms

In this section, we briefly outline the adaptation by Ekerå and Gärtner [EG24] of Regev's factoring algorithm [Reg23] to the discrete logarithm problem modulo prime p, and highlight that our error tolerance result (specifically, Lemma 4.4) also directly applies to their algorithm.

The algorithm follows a very similar blueprint to [Reg23]. It runs a quantum circuit of size $O(n^{1/2} \cdot G + n^{3/2})$ (where G is the size of an integer multiplication circuit as in Theorem 1.1) for $O(n^{1/2})$ iterations to obtain several samples from the dual of some lattice. Because of the large number of runs of the quantum circuit, this algorithm also stands to benefit in principle from a classical postprocessing procedure that tolerates a constant fraction of unsuccessful runs.

Let the inputs for the discrete logarithm problem be a generator g and target value x. Let $b_1, \ldots, b_{d-2}$ be some small integers as in the factoring algorithm by [Reg23]. Then Ekerå and Gärtner [EG24] construct the following lattice, for a generator g and target value x:

$$\mathcal{L}_{x,g} = \left\{ (z_1, \ldots, z_d) \in \mathbb{Z}^d \mid x^{z_{d-1}} g^{z_d} \prod_{i=1}^{d-2} b_i^{z_i} \equiv 1 \pmod{p} \right\}.$$

For this lattice, they make a similar number-theoretic assumption to that proposed by [Reg23]:

Conjecture E.1. *There exists a basis for $\mathcal{L}_{x,g}$, where all basis elements have ℓ_2 norm at most $T = 2^{C\sqrt{n}}$, for some given constant $C > 0$.*

They then argue via Lemma 3.4 that Regev's classical postprocessing procedure [Reg23] will find a collection of vectors in $\mathcal{L}_{x,g}$ that generates all elements in $\mathcal{L}_{x,g}$ of magnitude at most T. Under Conjecture E.1, this must mean they generate $\mathcal{L}_{x,g}$ itself, hence we have a basis for $\mathcal{L}_{x,g}$ [EG24]. Now one can directly solve to find elements $(z_1, \ldots, z_d) \in \mathcal{L}_{x,g}$ such that $z_1 = z_2 = \ldots = z_{d-2} = 0$ and thus recover the discrete logarithm [EG24].

By plugging in the postprocessing procedure in our Lemma 4.4 in place of Lemma 3.4, one can readily obtain a result analogous to Theorem 4.5 for the discrete logarithm problem modulo p.

F Calculating Parameters

F.1 Justifying the Choice of Parameters in [Reg23]

Plugging everything into the inequality stated in Lemma 3.4 implies that we require:

$$\begin{aligned}
&(2d+4)^{1/2}\cdot 2^{d+2}\cdot(d+5)^{1/2}\cdot T < \delta^{-1}(4\cdot 2^n)^{-1/(d+4)}/6\\
\Leftrightarrow\ &(2d+4)^{1/2}\cdot 2^{d+2}\cdot(d+5)^{1/2}\cdot 2^{C\sqrt{n}} < 2^{(A-o(1))n/d}\cdot(4\cdot 2^n)^{-1/(d+4)}/6\\
\Leftrightarrow\ &(A-o(1))n/d > \log_2(6d^{1/2}\cdot(d+2)^{1/2}\cdot 2^{d+2}\cdot(d+5)^{1/2}\cdot 2^{C\sqrt{n}}\cdot 2^{(n+2)/(d+4)})\\
&\qquad = o(\sqrt{n}) + C\sqrt{n} + (d+2) + \frac{n+2}{d+4}\\
&\qquad = o(\sqrt{n}) + C\sqrt{n} + \sqrt{n} + \sqrt{n}\\
&\qquad = (C+2+o(1))\sqrt{n},
\end{aligned}$$

thus taking $A \geq C+2+o(1)$ is sufficient.

F.2 Justifying the Choice of Parameters in Theorem 4.5

Since $d=\sqrt{n}$, we can plug in $T=2^{Cd}$ and also use the bound $\det\mathcal{L}<2^n$ and set $\delta=2^{(-A+o(1))d}$. With these parameters, the special inequality becomes:

$$\begin{aligned}
&(\gamma d+d)^{1/2}\cdot 2^{(\gamma+1)d/2}\cdot(\gamma d+1)^{1/2}\cdot T < \delta^{-1}\cdot(4\det\mathcal{L})^{-1/(\gamma d)}\cdot 2^{-\alpha/\gamma}/6\\
\Leftarrow\ &(\gamma d+d)^{1/2}\cdot 2^{(\gamma+1)d/2}\cdot(\gamma d+1)^{1/2}\cdot 2^{Cd} < 2^{(A-o(1))d}\cdot(4\cdot 2^n)^{-1/(\gamma d)}\cdot 2^{-\alpha/\gamma}/6\\
&\qquad = 2^{(A-1/\gamma-o(1))d}\\
\Leftrightarrow\ &2^{(C+\gamma/2+1/2+o(1))d} < 2^{(A-1/\gamma-o(1))d},
\end{aligned}$$

which holds for $A \geq C+\frac{\gamma^2+\gamma+2}{2\gamma}+o(1)$.

G Proof of Lemma 4.3

Let us temporarily fix coefficients a_i such that at least one is nonzero. Let us also fix a vector $v\in\Lambda^*/\mathbb{Z}^d$ such that $d(\sum_{i=1}^k a_i\epsilon_i, v)\leq 2^{(-2A+\alpha-\gamma+3+o(1))n/(2d)}$. We will take a union bound over all such choices at the end.

Since at least one a_i is nonzero and the a_i's are all integers, it is easy to see that $\sum_{i=1}^k a_i\epsilon_i$ will be uniformly distributed over $\mathbb{R}^d/\mathbb{Z}^d$. Hence $\sum_{i=1}^k a_i\epsilon_i - v$ will also be a uniformly distributed sample from $\mathbb{R}^d/\mathbb{Z}^d$.

We want to bound the probability that the norm of this sample on the torus is at most $2^{(-2A+\alpha-\gamma+3+o(1))n/(2d)}$. This is at most the probability that each coordinate of our sample is that close to an integer, which is at most:

$$\left(2\cdot 2^{(-2A+\alpha-\gamma+3+o(1))n/(2d)}\right)^d = 2^{(-2A+\alpha-\gamma+3+o(1))n/2}.$$

Finally, we take our union bound over the following:

- The choice of $v \in \Lambda^*/\mathbb{Z}^d$. There are at most $|\Lambda^*/\mathbb{Z}^d| = \det \Lambda \leq N < 2^n$ such choices.
- The choice of the a_i's. There are at most $2^{(\alpha-\gamma+3+o(1))n/(2d)\cdot k} \leq 2^{(\alpha(\alpha-\gamma+3)+o(1))n/2}$ such choices.

Hence the probability of problematic integers $a_1, \ldots, a_k$ existing is at most:

$$2^n \cdot 2^{(\alpha(\alpha-\gamma+3)+o(1))n/2} \cdot 2^{(-2A+\alpha-\gamma+3+o(1))n/2} = 2^{(-2A+(\alpha+1)(\alpha-\gamma+3)+2)n/2},$$

which is negligible provided $A > \frac{(\alpha+1)(\alpha-\gamma+3)+2}{2}$. □

H Proof of Corollary 5.3

We write the full computation out below for clarity:

$$\begin{aligned}
&|\phi\rangle |b\rangle |b^{-1}\rangle |g\rangle |0^{\max(S,S_1,S_2)+n}\rangle \\
&\xrightarrow{U_1} |\phi_1\rangle |a\rangle |b\rangle |b^{-1}\rangle |g\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{\text{mult}} |\phi_1\rangle |a\rangle |b\rangle |b^{-1}\rangle |g+ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{U_2\circ U_1^{-1}} |\phi_2\rangle |a^{-1}\rangle |b\rangle |b^{-1}\rangle |g+ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{\text{mult}} |\phi_2\rangle |a^{-1}\rangle |b-(a^{-1}\cdot(g+ab))\rangle |b^{-1}\rangle |g+ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\qquad = |\phi_2\rangle |a^{-1}\rangle |-a^{-1}g\rangle |b^{-1}\rangle |g+ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{U_1\circ U_2^{-1}} |\phi_1\rangle |a\rangle |-a^{-1}g\rangle |b^{-1}\rangle |g+ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{\text{mult}} |\phi_1\rangle |a\rangle |-a^{-1}g\rangle |b^{-1}\rangle |(g+ab)+a\cdot(-a^{-1}g)\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\qquad = |\phi_1\rangle |a\rangle |-a^{-1}g\rangle |b^{-1}\rangle |ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{U_2\circ U_1^{-1}} |\phi_2\rangle |a^{-1}\rangle |-a^{-1}g\rangle |b^{-1}\rangle |ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{\text{mult}} |\phi_2\rangle |a^{-1}\rangle |-a^{-1}g+a^{-1}b^{-1}\rangle |b^{-1}\rangle |ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{U_1\circ U_2^{-1}} |\phi_1\rangle |a\rangle |-a^{-1}g+a^{-1}b^{-1}\rangle |b^{-1}\rangle |ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{\text{mult}} |\phi_1\rangle |a\rangle |-a^{-1}g+a^{-1}b^{-1}\rangle |b^{-1}-(a\cdot(-a^{-1}g+a^{-1}b^{-1}))\rangle |ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\qquad = |\phi_1\rangle |a\rangle |-a^{-1}g+a^{-1}b^{-1}\rangle |g\rangle |ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{U_2\circ U_1^{-1}} |\phi_2\rangle |a^{-1}\rangle |-a^{-1}g+a^{-1}b^{-1}\rangle |g\rangle |ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{\text{mult}} |\phi_2\rangle |a^{-1}\rangle |(-a^{-1}g+a^{-1}b^{-1})+a^{-1}\cdot g\rangle |g\rangle |ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\qquad = |\phi_2\rangle |a^{-1}\rangle |(ab)^{-1}\rangle |g\rangle |ab\rangle |0^{\max(S,S_1,S_2)}\rangle \\
&\xrightarrow{U_2^{-1}} |\phi\rangle |(ab)^{-1}\rangle |g\rangle |ab\rangle |0^{\max(S,S_1,S_2)+n}\rangle \\
&\to |\phi\rangle |ab\rangle |(ab)^{-1}\rangle |g\rangle |0^{\max(S,S_1,S_2)+n}\rangle .
\end{aligned}$$

There are a constant number of calls to our multiplication circuit and the unitary circuits U_1 and U_2. Finally, some n-bit registers are swapped at the end. This implies the stated gate complexity, completing the proof of the corollary. □

I Proof of Correctness in Lemma 5.4

It remains to show the final claim made in the proof of Lemma 5.4 by induction. First we check the base case i.e. the first two rounds where $j = K, K-1$. In round K, (x_1, x_2) evolves as follows: $(1,1) \to (1,1) \to (c_K, 1) \to (1, c_K)$. Then in round $K-1$, it evolves as follows: $(1, c_K) \to (c_K, c_K) \to (c_{K-1}c_K, c_K) \to (c_K, c_{K-1}c_K)$. This is consistent with our claim for $j = K-1$.

For the inductive step, assume the current round is j, and the previous round (indexed by $j+1$) has ended according to our claim. The current state is hence:

$$(x_1, x_2) = \left(\prod_{i=j+2}^{K} c_i^{F_{i-j-1}}, \prod_{i=j+1}^{K} c_i^{F_{i-j}} \right).$$

After the first update, x_1 becomes:

$$\begin{aligned} \prod_{i=j+2}^{K} c_i^{F_{i-j-1}} \cdot \prod_{i=j+1}^{K} c_i^{F_{i-j}} &= c_{j+1}^{F_1} \cdot \prod_{i=j+2}^{K} c_i^{F_{i-j-1}+F_{i-j}} \\ &= c_{j+1} \cdot \prod_{i=j+2}^{K} c_i^{F_{i-j+1}} \\ &= \prod_{i=j+1}^{K} c_i^{F_{i-j+1}}. \end{aligned}$$

After the second update, it becomes:

$$c_j \cdot \prod_{i=j+1}^{K} c_i^{F_{i-j+1}} = \prod_{i=j}^{K} c_i^{F_{i-j+1}}.$$

Swapping the registers now gives us the state

$$(x_1, x_2) = \left(\prod_{i=j+1}^{K} c_i^{F_{i-j}}, \prod_{i=j}^{K} c_i^{F_{i-j+1}} \right),$$

completing our induction.

After the $j = 1$ stage, our state is hence $(\prod_{i=2}^{K} c_i^{F_{i-1}}, \prod_{i=1}^{K} c_i^{F_i})$, as desired. □

J Decomposing Positive Integers as a Sum of Fibonacci Numbers

It was shown by [Zec72] that any positive integer has a unique decomposition as a sum of Fibonacci numbers, if we enforce that no two of the Fibonacci numbers should be consecutive. We only need a weaker property, which we restate and prove here:

Lemma J.1. *Consider the following algorithm, that takes as input a non-negative integer $t_0 < F_{k+1}$. Initialize $t \leftarrow t_0$ and S to be the empty set. Now for $j = k, k-1, \ldots, 1$, check whether $t \geq F_j$. If it is, update $t \leftarrow t - F_j$ and add F_j to S.*

Then at the end of the algorithm, we will have $t = 0$ and the elements of S adding to t_0.

Proof. First, observe that the algorithm clearly ensures that $t \geq 0$ at all times, and that t and the elements of S together add to t_0. Hence it suffices to show that at the end of the algorithm, we will have $t = 0$.

We do this by strong induction on k. The base case $k = 1$ follows trivially since $t < F_2 = 1$ forces $t = 0$, so we are already done. We also consider the base case $k = 2$; in this case we have $t < F_3 = 2$. If $t = 0$ then we are already done. Else we have $t = 1 = F_2$. The algorithm will hence add F_2 to S and t will become 0.

Now for the inductive step, consider $k > 2$. We have two cases:

- If $t < F_k$, then in the $j = k$ round nothing will happen, and we simply reduce to the $k - 1$ case.
- If $t \geq F_k$, then in the $j = k$ round t will be replaced by $t - F_k < F_{k+1} - F_k = F_{k-1}$. Hence nothing will happen in the $j = k - 1$ round, and at this point we have reduced to the $k - 2$ case.

Either way, the conclusion follows by induction. □