The Jacobi Factoring Circuit:

Quantum Factoring with Near-Linear Gates and Sublinear Space and Depth

Gregory D. Kahanamoku-Meyer* MIT

Seyoon Ragavan† MIT

Vinod Vaikuntanathan‡ MIT

Katherine Van Kirk§ Harvard

July 2, 2026

Abstract

We present a compact quantum circuit for factoring a large class of integers, including some whose classical hardness is expected to be equivalent to RSA (but not including RSA integers themselves). Most notably, we factor n-bit integers of the form P^2Q with $\log Q = \Theta(n^a)$ for $a \in (2/3, 1)$ in space and depth sublinear in n (specifically, $\widetilde{O}(\log Q)$) using $\widetilde{O}(n)$ quantum gates; for these integers, no known classical algorithms exploit the relatively small size of Q to run asymptotically faster than general-purpose factoring algorithms. To our knowledge, this is the first polynomial-time circuit to achieve sublinear qubit count for a classically-hard factoring problem.

Our circuit builds on the quantum algorithm for squarefree decomposition discovered by Li, Peng, Du, and Suter (Nature Scientific Reports 2012), which relies on computing the Jacobi symbol in quantum superposition. The technical core of our contribution is a new space-efficient quantum algorithm to compute the Jacobi symbol of A mod B, in the regime where B is classical and much larger than A. Our circuit for computing the Jacobi symbol generalizes to related problems such as computing the greatest common divisor and modular inverses, and thus could be of independent interest.

*Email: gkm@mit.edu. Supported by U.S. DoE Co-design Center for Quantum Advantage (C^2QA) DE-SC0012704.

†Email: sragavan@mit.edu. Supported by NSF CNS-2154149, a Simons Investigator Award, the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-25-C-0300, and Amazon Research Awards. 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 Defense Advanced Research Projects Agency (DARPA).

‡Email: vinodv@mit.edu. Supported by NSF CNS-2154149 and a Simons Investigator Award.

§Email: kvankirk@g.harvard.edu. Supported by the Fannie and John Hertz Foundation and an NDSEG fellowship.

Contents

1 Introduction **1**

1.1 The LPDS Circuit for Squarefree Decomposition 4

1.2 Technical Overview 6

2 Preliminaries **9**

2.1 Notation 9

2.2 The Jacobi Symbol 9

2.3 Computational Number Theory 11

2.4 Sums of Phases 13

2.4.1 Basic Lemmas 13

2.4.2 Gauss Sums 14

3 Factoring Squarefull Integers **15**

4 Algorithm for Computing Jacobi Symbols **22**

4.1 Abstract Construction 22

4.2 Implications: Factoring Certain Integers in Sublinear Space and Depth 30

5 Completely Factoring Integers with Distinct Exponents in their Prime Factorization **30**

1 Introduction

Shor's discovery of a polynomial-time quantum algorithm for factoring numbers [Sho97] jump-started the field of quantum computation. However, despite decades of intense research and development in quantum algorithms, quantum error correction, and quantum hardware, quantum factoring circuits remain out of reach for current devices.

The difficulty is many-fold; a primary issue is that that the quantum circuits for factoring are still rather large, in terms of gate count, space complexity, and gate depth. For example, Shor's algorithm seems to require quantum circuits of size $\tilde{O}(n^2)$ to factor n-bit numbers [Sho97] (where the notation $\tilde{O}$ hides factors poly-logarithmic in n); a recent improvement by Regev has reduced the asymptotic gate count to $\tilde{O}(n^{3/2})$ per run [Reg25], but the practical cost of achieving this improved scaling seems rather large [EG24b]. Much effort has been applied to reducing the space-complexity of these circuits, for both Shor [BCDP96, VBE96, Sei01, Cop02, CW00, Bea03, TK06, Zal06, EH17, Gid17, HRS17, Gid19, GE21, KMY24] and Regev's algorithms [RV24, EG24a], achieving space as low as $\widetilde{O}(n)$ or even $O(n)$ qubits. Particularly notable is a recent work which reduced the space cost of Shor's algorithm to $n/2 + o(n)$ qubits [CFS24]. Yet there seems to be no fundamental obstacle preventing these costs from being improved further, and indeed if we are to have any hope of factoring classically-intractable integers on quantum computers in the near- or medium-term, it will be necessary to do so. Thus we arrive at two questions that are the focus of this paper:

Are there quantum circuits for factoring with (near-)linear gate count?
Could these be implemented with sub-linear space and depth?

In a nutshell, our main contribution is to present the *Jacobi factoring circuit*, a quantum circuit for factoring a large class of integers for which efficient classical algorithms are not known. Our circuit completely factors any number N whose prime decomposition has distinct exponents, and finds at least one non-trivial factor if any exponent is ≥ 2 (i.e. N is divisible by the square of some prime). Notably, this excludes RSA composites $N = PQ$ that are a product of two primes. We state below the special case of $N = P^2Q$ with P and Q prime.

Theorem 1.1 (Informal, see Corollary 4.7 for formal statement)**.** *There is a quantum circuit that factors any n-bit integer $N = P^2Q$ (with P and Q prime, and $Q < 2^m$ for some m) with $\widetilde{O}(n)$ gates, $\widetilde{O}(m)$ qubits, and $\widetilde{O}(m + n/m)$ depth.*

The space and depth complexity are sublinear when $m = \Theta(n^a)$ with $a \in (0, 1)$. Algorithms to factor numbers of this form have been extensively studied in the cryptography and computational number theory literature [PO96, BDH99, CJLN09, CL09, May10, CFRZ16, HH22, Mul24]; motivated in part by the fact that this problem's hardness has been used as the basis of several cryptosystems [Oka90, OU98, PT00, Tak98, SS06]. Roughly speaking, to classically factor $N = P^2Q$ where Q is the smaller of the two numbers, we have two choices. Either employ a class of special-purpose factoring algorithms along the lines of Lenstra's elliptic curve method [Len87], which run in time $\exp(\tilde{O}(\sqrt{\log Q}))$; or use the fastest general-purpose factoring algorithm, namely the number field sieve [Pol93, LLMP90, BLP93], which runs in time $\exp(\tilde{O}((\log N)^{1/3}))$. Which one is faster depends on how small Q is relative to N. As long as $\log Q = \widetilde{\Omega}((\log N)^{2/3})$, there are no known *classical* algorithms that exploit the special structure in N to factor faster than general-purpose

factoring algorithms. We refer the reader to Section 2.3 for more in-depth discussion on special-purpose factoring.

Before proceeding further, let us mention that the other barriers to realizing integer factorization on a quantum computer come from the concrete costs of factoring circuits; from the overhead due to quantum error-correction [GE21]; and from the difficulty in building quantum hardware [AAB+19], none of which we address in this paper. We do note, however, that the Jacobi factoring circuit, appropriately instantiated, seems friendly enough to admit a concretely small realization that factors, say, 2048-bit integers of the form stated in Theorem 1.1, although we leave an exploration of the circuit's concrete costs to future work.

The Jacobi factoring circuit builds on the remarkable, but apparently little known, work of Li, Peng, Du and Suter [LPDS12] who constructed a quantum circuit to compute the squarefree decomposition of an integer. That is, given as input a positive integer N, find the unique A and B such that $N = A^2B$ and B is not divisible by the square of any integer (greater than 1). The quantum part of the LPDS circuit computes a Jacobi symbol mod N, followed by a quantum Fourier transform mod N. Using the algorithm of Hales and Hallgren [HH00], the quantum Fourier transform mod N can be computed (approximately) with near-linear size quantum circuits. We observe that using Schönhage's GCD algorithm [Sch71, TY90, BS96, Möl08], Jacobi symbols can be computed in near-linear time as well. Overall, this gives a near-linear size (and also near-linear space and depth) quantum circuit for squarefree decomposition which, in particular, factors numbers $N = P^2Q$ where P, Q are prime numbers.

Our Contributions. Building on the aforementioned work of Li, Peng, Du and Suter [LPDS12], we show the following:

- Our first contribution, presented in Section 3, is a new analysis of [LPDS12] that is *necessary* to get our final result. Jumping ahead a bit, it allows the quantum circuit to use a superposition of numbers from a potentially much smaller range, e.g. to factor $N = P^2Q$, one can use a superposition of numbers from 1 to $\mathsf{poly}(Q)$ rather than 1 to $\mathsf{poly}(N)$ as in [LPDS12]. (Hence the number of qubits required for the initial superposition will be $\log(\mathsf{poly}(Q)) = O(\log Q)$.)

- Our second and main technical contribution, presented in Section 4, is the construction of an efficient quantum circuit to compute the Jacobi symbol $\left(\frac{A}{B}\right)$ in near-linear size and sublinear space and depth, when $\log A \ll \log B$, and A could be in superposition but B is classical. In particular, our circuit achieves qubit count $\widetilde{O}(\log A)$ and gate count $\widetilde{O}(\log B)$, parallelized into depth at most $\widetilde{O}(\log B/\log A + \log A)$. Combined with our first contribution, this yields a factoring circuit for P^2Q with essentially the same gate count as [LPDS12] but using smaller space and depth when $\log Q \ll \log P$.

  We believe our circuit design is of independent interest as it can be readily adapted to solve other problems of a similar nature, e.g. computing the greatest common divisor $\gcd(A, B)$ with the same efficiency.

- Our final contribution, presented in Section 5, is the observation that an algorithm for squarefree decomposition suffices to completely factor a general class of integers of inverse-polynomial density, namely any integer whose prime factorization has distinct exponents. A similar observation is well-known in the context of factoring polynomials [Yun76], and the high-level ideas are similar between the two settings.

On Special-Purpose Classical and Quantum Factoring. Our result can be seen to complement the classical factoring algorithms, e.g. [Len87, Mul24] that exploit various types of structure. The result by [LPDS12] takes a first step in this direction by showing that some integers with special structure (e.g. $N = P^2Q$ with P, Q prime) become polynomially easier to factor quantumly. Our result builds on this, demonstrating that these integers can be quantumly factored even more easily (i.e. in much lower space and depth) if $\log Q \ll \log N$. We emphasize that, as long as $\log Q \geq \widetilde{\Omega}((\log N)^{2/3})$, this structure cannot be exploited by any known special-purpose classical factoring algorithms (we discuss this more in Section 2.3).

In contrast, prior quantum factoring algorithms [Sho97, Reg25] do not seem to benefit *polynomially* from any such structure. Remarkable algorithms by [EH17, CFS24] that achieve constant-factor improvements over Shor's original construction depend upon the input being an RSA integer (the product of two primes of equal bit length), but no prior algorithms have better asymptotic scaling depending on the structure of the input. A related, and important, open question is whether our algorithm can be leveraged or extended to factor integers in general (we note that it would suffice to devise a way to factor squarefree integers; see Sections 3 and 5 for details).

Another important future direction is classical cryptanalysis for factoring integers of the form P^2Q where Q is much smaller than P, since this is the regime in which we get sublinear space. To the best of our knowledge, existing algorithms [Len87, BDH99, Mul24] do not offer any significant improvements in this regime, but this is not a regime that was previously of much practical interest, so we leave further investigation along these lines to future work.

A More Efficient Proof of Quantumness. Recent excitement has centered on *efficiently-verifiable proofs of quantumness*, which are protocols by which a single untrusted quantum device can demonstrate its quantum capability to a skeptical polynomial-time classical verifier [BCM+21, BKVV20, KCVY21, YZ22, KLVY23, MY23, AMMW24, AZ24, Mil24]. The Jacobi factoring circuit presented in this work immediately yields the first factoring-based proof of quantumness with sublinear space complexity (see Table 1). Existing proofs of quantumness based on factoring broadly fall into two categories: factoring algorithms, which straightforwardly demonstrate their quantum capability by finding the factors; and interactive protocols, which do not actually factor the number, but instead perform a task that for any classical algorithm is provably as hard as factoring. We address each of these in turn:

- *Factoring algorithms:* Shor's algorithm for factoring [Sho97], when implemented with a low-depth quantum multiplication circuit [NZLS23], costs $\widetilde{O}(n^2)$ gates, $\widetilde{O}(n)$ qubits, and $\widetilde{O}(n)$ depth.[1] Regev's recent improved factoring algorithm [Reg25], together with the optimizations of [RV24] (and using the same low-depth multiplier), can be implemented in $\widetilde{O}(n^{1.5})$ gates, $\widetilde{O}(n)$ qubits, and $\widetilde{O}(n^{0.5})$ depth. Note, however, that this is the gate count per run, and as $O(n^{0.5})$ runs are required, Regev does not improve total gate count across runs. The previously proposed Jacobi factoring circuit [LPDS12], together with the algorithm by [Sch71] for computing Jacobi symbols, uses $\widetilde{O}(n)$ gates, space, and depth.

  In contrast, if we instantiate our construction with an integer $N = P^2Q$ where $\log Q = \widetilde{\Theta}((\log N)^{2/3})$, our circuit uses $\widetilde{O}(n)$ gates, $\widetilde{O}(n^{2/3})$ depth, and $\widetilde{O}(n^{2/3})$ qubits. In terms of the product of qubit count with either gates or depth, this outperforms all other factoring algorithms described here.

[1] There also exist log-depth implementations of Shor's algorithm [CW00], but they come at the cost of far worse gate and qubit counts. We include asymptotics for this circuit in Table 1 for completeness.

- *Interactive protocols* that do not factor: the relevant protocols here are those based on trapdoor claw-free functions (TCFs), specifically instantiated with Rabin's function $f(x) = x^2 \bmod N$ as introduced in [KCVY21]. Evaluating this function requires performing just a single multiplication, and thus can be implemented with a quantum circuit of $\widetilde{O}(n)$ gates, $\mathsf{polylog}(n)$ depth, and $\widetilde{O}(n)$ qubits. While the low depth is appealing, the main obstacle is the qubit count, which is outperformed substantially by the Jacobi factoring circuit. Furthermore, the protocol of [KCVY21] is interactive, requiring the quantum computer to maintain coherence throughout several rounds of measurement of subsets of the qubits and communication with the verifier. Indeed, that protocol cannot be run in an "offline" setting, where a classical verifier publishes a challenge publicly and provides no further data to any particular prover. Interactive protocols are also somewhat less satisfying as a proof of quantum computational power, because the prover is not actually solving a computational problem—instead, interaction allows the prover show it can make measurements in anticommuting bases, which is not possible for a classical algorithm. There do exist TCF-based protocols which are non-interactive; their classical hardness either relies on quantum access to random oracles ([BKVV20], see [CGH04, KM15] for discussion of the random oracle heuristic) or computational problems other than factoring [AGGM24].

For completeness we note that, when the quantum circuit costs are expressed as a function of the best-known classical time cost for the same problem, our result does not asymptotically outperform certain proofs of quantumness based the hardness of problems other than factoring — simply because the classical hardness of those problems grows much more rapidly. Consider, for example, applying Shor's algorithm to the elliptic curve discrete logarithm problem (ECDLP) [Sho97, HJN+20]. Although the standard quantum circuit to solve ECDLP requires at least linear space and depth in the size of the input — which is worse than the circuits we present in this work — this is outweighed by the fact that integer factorization admits sub-exponential time classical algorithms (namely $\exp(\widetilde{O}(n^{1/3}))$), while to the best of our knowledge ECDLP does not. Thus, if we want to work with a problem that takes time T to solve classically, it would suffice to set $n = O(\log T)$ in the case of ECDLP, whereas for factoring we would need to set $n = \widetilde{O}((\log T)^3)$. However, in practice the constant factors for factoring circuits seem to be dramatically better than those for the ECDLP problem, with the constant multiplying the leading-order term even being less than 1 in some cases [GE21].

We proceed to describe the quantum circuit of [LPDS12] and then our technical contributions in more detail.

1.1 The LPDS Circuit for Squarefree Decomposition

In a beautiful work from a decade ago, Li, Peng, Du and Suter [LPDS12] showed a quantum circuit to compute the squarefree decomposition of an integer. That is, let $N = A^2B$ where B is not divisible by the square of any integer (greater than 1) denote the unique squarefree decomposition of N. Given N, computing B seems classically hard in general; indeed, it is at least as hard as factoring integers of the form $N = P^2Q$ where P and Q are primes. The squarefree decomposition problem has received much attention from the computational number-theory community [PO96, BDH99, CJLN09, CL09, May10, CFRZ16, HH22, Mul24], in part due to its applications in cryptography [Oka90, OU98, PT00, Tak98, SS06], and it is at the core of other important problems such as computing the ring of integers of a number field [BL94] and the endomorphism ring of an elliptic curve over a finite field [BS11].

Protocol	Cost (up to polylog factors)		
	Gates	Depth	Qubits
Shor [Sho97]	n^2	n	n
Log-depth Shor [CW00]	n^5	$\log n$	n^5
Regev [Reg25, RV24]	$n^{3/2}$	$n^{1/2}$	n
$x^2 \bmod N$ [KCVY21]†	n	$\log^2 n$	n
Squarefree decomposition [LPDS12, Sch71]	n	n	n
This work	$\boldsymbol{n}$	$\boldsymbol{n^{2/3}}$	$\boldsymbol{n^{2/3}}$

Table 1: **Asymptotic cost of various proofs of quantumness based on the hardness of factoring n-bit integers.** We omit constant and poly-logarithmic factors throughout for clarity. For all algorithms which use black-box multiplication, we assume the use of a parallelized circuit for Schönhage-Strassen multiplication [NZLS23]. We use † to denote the fact that [KCVY21] is an interactive protocol in which the quantum computer is not required to actually factor the number.

The starting point of [LPDS12] is the observation that when $N = P^2Q$, the Jacobi symbol of x mod N depends essentially[2] only on x mod Q. Indeed, if x and N are relatively prime,

$$\left(\frac{x}{N}\right) = \left(\frac{x}{P}\right)^2 \left(\frac{x}{Q}\right) = \left(\frac{x}{Q}\right)$$

since $\left(\frac{x}{P}\right) \in \{\pm 1\}$. Thus, the Jacobi symbol of x mod N is periodic modulo the secret factor Q.

With quantum period finding in mind, this naturally suggests the following procedure: (1) start with a uniform superposition over all x mod N; (2) compute and measure the Jacobi symbol $\left(\frac{x}{N}\right)$; and (3) use Shor's period-finding procedure [Sho97] to recover Q. The apparent obstacle is that once we apply a phase of $\left(\frac{x}{N}\right)$, we will end up not with one periodic signal modulo Q but a *superposition* of several periodic signals modulo Q.

One approach to circumvent this obstacle would be the following: instead of measuring one Jacobi symbol, we could measure multiple Jacobi symbols $\left(\frac{x}{N}\right), \left(\frac{x+1}{N}\right), \ldots, \left(\frac{x+k}{N}\right)$ for a large enough k so as to (hopefully) uniquely determine the value of x mod Q. (Intuitively, each one of these Jacobi symbols should give an "independent" piece of information about the value of x mod Q, so measuring enough of them should determine x mod Q.) It turns out that $k = \mathsf{poly}(\log Q)$ likely suffices (see the Boneh-Lipton conjecture [BL96, CW24]). Thus measuring the function $\mathsf{manyJac}_{N,k}(x) = \left(\frac{x}{N}\right), \left(\frac{x+1}{N}\right), \ldots, \left(\frac{x+k}{N}\right)$ on the uniform superposition gives us

$$\frac{1}{\sqrt{N}} \cdot \sum_{x \in [0, N-1]} |x\rangle \xrightarrow{\text{measure } \mathsf{manyJac}_{N,k}} \frac{1}{P} \cdot \sum_{j \in [0, P^2-1]} |x_0 + jQ\rangle \xrightarrow{\text{QFT mod } N} \frac{1}{\sqrt{Q}} \sum_{j \in [0, Q-1]} e^{-2\pi i x_0 j/Q} \left| \frac{jN}{Q} \right\rangle \quad (1)$$

for some $x_0 \in [0, Q-1]$. Now, measuring gives us an integer multiple of $N/Q = P^2$ from which it is not hard to read off P^2 and therefore P.

[2] We say "essentially" because this is subject to the minor constraint that x and N need to be relatively prime.

This would, however, result in a rather large circuit: to uniquely fix x mod Q, one would certainly need to compute at least $\Omega(\log Q)$ Jacobi symbols (and perhaps even a larger $\mathsf{poly}(\log Q)$ [BL96, CW24]). The key result of [LPDS12] is that just computing and measuring a single Jacobi symbol already suffices (even though we would be working with a superposition of periodic signals[3]). Even better, instead of measuring the Jacobi symbol we can simply apply a phase equal to the Jacobi symbol $\left(\frac{x}{N}\right)$. The effect of this is to eliminate any amplitude placed by the post-QFT state on $|0\rangle$ (which is not helpful for factoring N). Indeed, the QFT of the signal after measuring a single Jacobi symbol, namely that of x mod N, will be very similar to the end result in Equation (1) except that each basis state will receive a sum of several amplitudes. In particular, if we apply a phase equal to the Jacobi symbol $\left(\frac{x}{N}\right)$[4] each non-zero basis state $|jN/Q\rangle$ will have absolute amplitude

$$\approx \frac{1}{Q} \cdot \left| \sum_{x_0 \in [1, Q-1]} \left(\frac{x_0}{Q}\right) \exp\left(-\frac{2\pi i x_0 j}{Q}\right) \right|.$$

By standard Gauss sum bounds (see Section 2.4.2 and Remark 3 for details), the summation is *lower-bounded* by $\Omega(\sqrt{Q})$, and so we know that each non-zero basis state will have amplitude $\Omega(1/\sqrt{Q})$. Since there are $Q-1$ such states, a measurement will give us a non-zero multiple of $N/Q = P^2$ with a constant probability (in fact, [LPDS12] shows that this probability is 1!).

We remark that [LPDS12] generalizes this method to obtain the squarefree decomposition of any N, not necessarily of the form P^2Q for prime P, Q. With this in mind, we now turn to an overview of our techniques.

1.2 Technical Overview

Section 3: A New Analysis of [LPDS12]. Our first contribution is a more careful analysis of the circuit by [LPDS12], wherein we show that it suffices to start with a superposition from 1 to $\mathsf{poly}(Q)$ rather than all the way to N. At a high level, this follows from combining two previous techniques. Our starting point is the analysis by Shor [Sho97] that shows that in order to find the period of a function with period $\leq Q_{\max}$, it suffices to take a superposition from 1 to $\mathsf{poly}(Q_{\max})$.

The reason this does not immediately suffice for our setting is that we are not working with one periodic signal; we would be working with a superposition of periodic signals corresponding to values $x_0 \in [0, Q-1]$ grouped according to their Jacobi symbol $\left(\frac{x_0}{Q}\right)$. To get around this, we combine elements of Shor's analysis [Sho97] with the Gauss sum analysis introduced by [LPDS12].

Section 4: Computing Jacobi Symbols in Sublinear Space and Depth. The computational bottleneck in the [LPDS12] factoring circuit is computing the Jacobi symbol $\left(\frac{x}{N}\right)$, where $x \in [0, N-1]$ is in superposition. This can be done in gates (and hence space/depth) $\widetilde{O}(n)$ [Sch71, TY90, BS96, Möl08], where n is the number of bits in N, which is a near-linear gate complexity and hence essentially tight.

[3] [HH00] provides a black-box algorithm for finding the period of "many-to-one" periodic functions like this, however it requires a super-constant number of calls to the Fourier sampling subroutine. The Gauss sum analysis of [LPDS12] (and that of the present work) provides much better efficiency, showing that just one iteration of Fourier sampling suffices to find the period with probability $\Omega(1)$.

[4] We once again assume here that the Jacobi symbol is always in $\{-1, 1\}$ even though it can occasionally also be 0; we will handle this more carefully in the relevant technical sections.

However, thanks to our first contribution, we need only compute $\left(\frac{x}{N}\right)$ for $x \leq \mathsf{poly}(Q)$; moreover, since N is classically known, there is the tantalizing possibility that the number of qubits could be pushed down to linear in $\log Q$ rather than $\log N$. We show that this is indeed the case (up to polylogarithmic factors), by constructing a quantum circuit that computes the Jacobi symbol using space $\widetilde{O}(m)$ qubits for any $m \geq \log Q$. Our circuit is also very efficient, achieving gate count $\widetilde{O}(n)$, parallelized into depth $\widetilde{O}(n/m + m)$.[5]

To explain our methods, let us revisit some well-known algorithms for computing the Jacobi symbol $\left(\frac{x}{N}\right)$. These algorithms also provide algorithms for computing GCDs and vice versa:

- The binary GCD algorithm [BS96] is often used in quantum algorithms due to its circuit-friendliness [RNSL17]. However, this will not be useful for our goals; the number of gates needed to compute $\left(\frac{x}{N}\right)$ is $O((\log x + \log N)^2)$ [BS96], which is quadratic in $\log N$ (rather than near-linear).
- The extended Euclidean algorithm relies on the observation that the Jacobi symbol $\left(\frac{a}{b}\right)$ is equal to $\left(\frac{a \bmod b}{b}\right)$, which together with quadratic reciprocity (property 7 of Theorem 2.4) allows one to rapidly reduce the size of the problem's inputs. Indeed, after just one step, the problem is reduced to the computation of the Jacobi symbol of two inputs of length $O(\log x)$. Nevertheless, due to that first step, this algorithm seems to require $\widetilde{O}(\log N)$ qubits.
- Finally, there is an algorithm due to Schönhage [Sch71, TY90, BS96, Möl08] that runs in $\widetilde{O}(\log N)$ gates, but does not come with any better guarantees on the space and depth.

We take an approach that, at a very high level, mimicks the extended Euclidean algorithm:

1. First, we reduce the computation of $\left(\frac{x}{N}\right)$ to some Jacobi computation $\left(\frac{a}{b}\right)$ between two inputs a, b of length $O(\log x)$.
2. We then compute $\left(\frac{a}{b}\right)$ using Schönhage's algorithm out-of-the-box, which only requires gates (and hence space/depth) $\widetilde{O}(\log x)$.

The challenge, and room for creativity, is in implementing step 1. To do this, we find a multiple kx of x such that both of the following are true: (a) $N - kx$ is divisible by 2^{n-m}; and (b) $kx < 2^n$. For now one should consider $m = \lceil \log x \rceil$; in some cases, one may choose $m \geq \lceil \log x \rceil$ to improve efficiency, as we discuss later. To be explicit, this allows us to compute the Jacobi symbol via the following chain of transformations:

$$\left(\frac{x}{N}\right) \to \left(\frac{N}{x}\right) \to \left(\frac{N - kx}{x}\right) \to \left(\frac{(N - kx)/2^{n-m}}{x}\right),$$

where each of the arrows follows from standard properties of the Jacobi symbol stated in Theorem 2.4, and the Jacobi symbol of the last expression is computed directly (step 2 above).

A conceptually simpler but concretely less direct and efficient variant[6] of our algorithm essentially follows the "extended Euclidean" blueprint: we use kx to compute $N \bmod x = \left[\left(\frac{N-kx}{2^{n-m}}\right) \cdot (2^{n-m} \bmod x)\right] \bmod x$. Then we can transform $\left(\frac{N}{x}\right) \to \left(\frac{N \bmod x}{x}\right)$ and finish from there using [Sch71]. In practice, this is

[5] If $\log Q < O((\log N)^{1/2})$, the depth and space cannot *both* be made to scale with $\log Q$ simultaneously, because the space-time volume (space times depth) is lower bounded by the gate count, and the gate count is lower bounded by $O(\log N)$. However, the parameters can be tuned to achieve a continuous tradeoff between the two, while maintaining a space-time product nearly linear in $\log N$.

[6] We thank Daniel J. Bernstein for suggesting this perspective on our algorithm.

unnecessary extra computation, so we proceed using our more specialized blueprint and do not bother with computing N mod x. One could of course attempt to directly compute N mod x as usual using long division, but it is not clear how to do this reversibly in low space and depth.

The key idea behind our quantum circuit for step 1 is to stream through the (classical) bits of N in blocks of size m starting with the lowest-order bits, matching each block of kx to the corresponding block of N (such that the difference $N - kx$ has trailing zeros). Sublinear quantum space is achieved via the observation that only the leading $O(m)$ bits of the running sum kx need to be stored quantumly at any given time, as all of the lower-order bits match the classical bits of N by design. Sublinear depth follows from the fact that the number of blocks is $O(n/m)$, and the desired operations on each block can be performed in a constant number of multiplications of depth $O(\text{polylog}(m))$ [NZLS23, SS71].

Our algorithm builds on Montgomery reduction [Mon85] and the binary GCD algorithm and can be thought of as a "reversed" variant of long division; long division starts from the most significant bit (MSB) and iterates towards the least significant bit (LSB) to find a multiple of x that agrees with N in the MSBs, whereas we start from the LSB and iterate towards the MSB to find a multiple of x that agrees with N in the LSBs. The benefit of proceeding in this reversed way is that uncomputing intermediate states now becomes easy, by way of a simple comparison that only depends on the MSBs of our intermediate state.

Section 5: Completely Factoring Special Integers. Finally, we present a black-box reduction implying that any algorithm for squarefree decomposition can be used to *completely factor* integers N with distinct exponents in their prime factorization; i.e. N that can be written as $p_1^{\alpha_1} \dots p_r^{\alpha_r}$ for distinct primes $p_1, \dots, p_r$ and distinct positive exponents $\alpha_1, \dots, \alpha_r$. Such integers have been studied before and are referred to as *special integers* [AM17] — in fact, these have even been proposed for use in cryptographic applications [Sch18].

A similar observation is well-known in the context of factoring polynomials; the motivation for this is that if a univariate polynomial f is divisible by the square of some (non-constant) polynomial, this can be detected easily by taking the GCD of f and its derivative f'. Yun [Yun76] shows that this can be extended to decompose any polynomial $f(x)$ into a factorization $f(x) = g_1(x)^{\alpha_1} \dots g_r(x)^{\alpha_r}$ where the g_i's are squarefree and pairwise coprime and the α_i's are distinct. Many polynomial factorization algorithms thus begin with this step as a subroutine.

Unsurprisingly, the algorithm in our setting bears some high-level similarity to Yun's algorithm [Yun76] and is based on a simple idea: using the algorithm for squarefree decomposition, we can recover

$$B = \prod_{i \in [r] \,:\, \alpha_i \text{ odd}} p_i.$$

Then let $i^* \in [r]$ be the index such that α_{i^*} is the smallest of the odd α_i. (If the α_i are all even, then N will be a perfect square and we can take its square root until at least one α_i is odd.) By dividing N by B as many times as possible, we obtain:

$$k = \prod_{i \in [r] \,:\, \alpha_i \text{ even}} p_i^{\alpha_i} \cdot \prod_{i \in [r] \,:\, \alpha_i \text{ odd}} p_i^{\alpha_i - \alpha_{i^*}}.$$

Now, because the α_i are all distinct, k will be divisible by every prime dividing B except p_{i^*}. Thus we can compute $B / \gcd(k, B) = p_{i^*}$, which is a prime divisor of N. We can now divide as many factors of p_{i^*} from N as possible, then recurse.

2 Preliminaries

2.1 Notation

Let $N < 2^n$ be an n-bit number that we wish to factor. We use $\mathsf{negl}(n)$ to denote any real-valued function $f(n)$ such that $|f(n)| = o(n^{-c})$ for all constants $c > 0$. For any positive integer k, let $\varphi(k)$ denote the number of positive integers in $[1, k]$ that are relatively prime to k. We will sometimes use the notation $A \mid B$ to indicate that the integer A divides the integer B.

We say that an integer B is *squarefree* if it is not divisible by any square (other than 1). Observe that any N has a unique representation of the form A^2B for some squarefree B; indeed, if N has prime factorization $\prod_{i=1}^{r} p_i^{a_i}$, then we must have $B = \prod_{i=1}^{r} p_i^{a_i \bmod 2}$ and $A = \prod_{i=1}^{r} p_i^{\lfloor a_i/2 \rfloor}$. When $A > 1$, we say that N is *squarefull*.

Throughout this paper, log will denote the base-2 logarithm. We use $\mathbb{Z}_N$ to denote the ring of integers mod N, and $\mathbb{Z}_N^*$ to denote the multiplicative group of invertible elements mod N. We will also use the following straightforward claim:

Proposition 2.1. *Let M, B be positive integers with $B > 1$ and let $j \in [0, B-1]$ be an integer. Then the number of integers $x \in [1, M]$ such that $x \equiv j \pmod{B}$ is exactly*

$$\left\lfloor \frac{M-j}{B} \right\rfloor - \left\lceil \frac{1-j}{B} \right\rceil + 1.$$

Proof. Writing $x = By + j$, we wish to find the number of integers y (not necessarily positive) such that:

$$\begin{aligned} & By + j \in [1, M] \\ \Leftrightarrow\ & y \in \left[\left\lceil \frac{1-j}{B} \right\rceil, \left\lfloor \frac{M-j}{B} \right\rfloor \right]. \end{aligned}$$

The conclusion now follows. □

2.2 The Jacobi Symbol

Here, we define the Jacobi symbol and state its relevant properties for our purposes. We follow the exposition in [BS96, Chapter 5]. The Legendre symbol is a well-known special case of the Jacobi symbol and our starting point:

Definition 2.2 (Legendre symbol)**.** *For an integer a and an odd prime p, define the Legendre symbol $\left(\frac{a}{p}\right)$ as follows:*

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{if } p \text{ divides } a; \\ 1, & \text{if } a \text{ is a (nonzero) quadratic residue mod } p; \\ -1, & \text{otherwise.} \end{cases}$$

The Jacobi symbol is most naturally defined in terms of the Legendre symbol:

Definition 2.3 (Jacobi symbol)**.** *Let a be an integer and b an odd positive integer with factorization $b = p_1^{e_1} \dots p_k^{e_k}$. Then define the Jacobi symbol $\left(\frac{a}{b}\right)$ as follows:*

$$\left(\frac{a}{b}\right) = \left(\frac{a}{p_1}\right)^{e_1} \left(\frac{a}{p_2}\right)^{e_2} \dots \left(\frac{a}{p_k}\right)^{e_k}.$$

At various points, we will use the notation $j_b(a) \in \{-1, 1\}$ to denote a value that is guaranteed to be $\left(\frac{a}{b}\right)$ when $\gcd(a, b) = 1$ and can be arbitrary otherwise (since as we will see below, in this case the Jacobi symbol would be 0).

The Jacobi symbol $\left(\frac{a}{b}\right)$ can be computed efficiently without knowing the factorization of b, via the following properties. They can be applied, for example, in the same manner as the extended Euclidean algorithm for the greatest common divisor; we will discuss quantum circuits for computing the Jacobi symbol in detail in Section 4.

Theorem 2.4 (Jacobi symbol properties)**.** *The Jacobi symbol has the following properties. (Recall that $\left(\frac{a}{n}\right)$ is only defined when n is an odd positive integer — although a could be even; thus in all of the below, it is assumed that m, n are both odd and positive.)*

1. *If $\gcd(a, n) > 1$, then $\left(\frac{a}{n}\right) = 0$. Otherwise, $\left(\frac{a}{n}\right) \in \{-1, 1\}$.*
2. $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$;
3. $\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right)$;
4. $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$ *if* $a \equiv b \bmod n$;
5. $\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}$;
6. $\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}$
7. *(Quadratic Reciprocity) If $\gcd(a, n) = 1$ and a is odd and positive, then*

$$\left(\frac{a}{n}\right)\left(\frac{n}{a}\right) = (-1)^{(a-1)(n-1)/4}.$$

Consequently, whenever a is odd and positive (perhaps having common factors with n), we will have

$$\left(\frac{a}{n}\right) = (-1)^{(a-1)(n-1)/4}\left(\frac{n}{a}\right).$$

(In the case that a, n have common factors, both sides will be 0.)

Corollary 2.5. *Let $\sigma \in \{-1, 1\}$. If N is odd and not a square, the number of integers $a \in [1, N]$ such that $\left(\frac{a}{N}\right) = \sigma$ is $\varphi(N)/2$.*

Proof. Since N is not a square, there exists a prime p_0 and odd integer e_0 such that $p_0^{e_0}$ divides N but $p_0^{e_0+1}$ does not. Let the other primes dividing N be $p_1, p_2, \dots, p_k$. Then let $b \in [1, N]$ be such that b is a quadratic non-residue mod p_0, and $b \equiv 1 \bmod p_i$ for all $i > 0$. Such b exists by the Chinese Remainder Theorem, and moreover we have for some exponents $e_0, \dots, e_k$ that $\left(\frac{b}{N}\right) = \left(\frac{b}{p_0}\right)^{e_0} \cdot \prod_{i=1}^{k}\left(\frac{b}{p_i}\right)^{e_i} = (-1)^{e_0} = -1$.

Then by property 2 of Theorem 2.4, $a \mapsto ab \bmod N$ provides a bijection between the elements of $\left\{x \in \mathbb{Z}_N^* : \left(\frac{x}{N}\right) = 1\right\}$ and $\left\{x \in \mathbb{Z}_N^* : \left(\frac{x}{N}\right) = -1\right\}$. It follows that these two sets have equal cardinality. They are disjoint, and in total they comprise $\varphi(N)$ elements by property 1 of Theorem 2.4, so the conclusion follows. □

2.3 Computational Number Theory

We survey the classical and quantum complexity of various computational number theory problems that are relevant to this work. Recall the well-known result that if we have a classical circuit that uses G gates to compute a function $f(x)$ of an input x, we can implement unitary computing $|x\rangle|0\rangle \mapsto |x\rangle|f(x)\rangle$ in $O(G)$ gates and $O(G)$ ancilla qubits [Ben73].

Arithmetic Operations. The fastest known classical circuits for n-bit integer multiplication use $O(n \log n)$ gates [HvdH21], and can be made quantum through standard reversibility techniques [Ben73, Ben89, LS90]. If space is a concern, one can use the multiplier due to [KMY24] which uses no ancilla qubits (i.e. it operates entirely in-place on the input and output registers), and has $O_\epsilon(n^{1+\epsilon})$ gates for any pre-specified $\epsilon > 0$. The depth of multiplication can be reduced to $O(\log^2 n)$ with the use of $\widetilde{O}(n)$ ancilla qubits, via a parallel quantum circuit for the Schönhage-Strassen algorithm which has gate count $\widetilde{O}(n)$ [NZLS23, SS71]. Via Newton iteration it is possible to perform division with the same complexity as multiplication (up to constant factors in the gate count and space, and a logarithmic factor in the depth) [Knu98].

Algorithms for Computing Jacobi Symbols and GCDs. The best-known algorithms for computing Jacobi symbols and GCDs of two n-bit integers are:

- The extended Euclidean algorithm, which can be classically done in $O(n^2)$ gates [BS96]. Moreover, this can be done quantumly in $O(n^2)$ gates while also keeping the space down to $O(n)$ qubits [PZ03].
- The binary GCD algorithm, which has the same asymptotic complexities as extended Euclidean both classically and quantumly [BS96, PZ03].

However, there is a faster divide-and-conquer algorithm which was conceived by Schönhage [Sch71] and [BS96, solution to exercise 5.52], and expounded Thull and Yap [TY90] and Möller [Möl08]. This algorithm runs in $\widetilde{O}(n)$ gates[7] (and hence at most that much space and depth).

Classical Algorithms for Factoring. The best-known classical algorithm for factoring arbitrary n-bit integers is the general number field sieve [Pol93, LLMP90, BLP93], which runs in heuristic time

$$\exp\left(O(n^{1/3}(\log n)^{2/3})\right).$$

However, there has also been extensive research towards generating faster classical algorithms, which exploit specific number theoretic structure present in the integer $N < 2^n$ being factored. For example, if $p < 2^{m'}$ is the smallest divisor of N, Lenstra's elliptic curve method [Len87] recovers p in heuristic time:

$$\mathsf{poly}(n) \cdot \exp\left(O\left(\sqrt{m' \log m'}\right)\right).$$

Furthermore, Mulder [Mul24] presents a classical algorithm specifically for squarefree decomposition, targeting the same structure as we do with the quantum algorithm in the present work. If $N = A^2 B$ with

[7] While these algorithms are usually formulated in the Turing machine model, they can be readily transformed into circuits at the expense of multiplicative $\mathsf{polylog}(n)$ overheads [PF79].

$B < 2^m$ squarefree, then Mulder's algorithm can recover A, B in heuristic time[8]

$$\mathsf{poly}(n) \cdot \exp\left(O\left(\sqrt{m \log m}\right)\right).$$

Thus, for n-bit integers of the form $N = p^2 q$ with p, q prime and $\log q = \widetilde{O}(n^{2/3})$, all known classical algorithms for factoring N require heuristic time

$$\exp\left(\widetilde{O}(n^{1/3})\right).$$

Since we consider algorithms for factorizing integers of the form $p^r q$ in the less-studied regime where q is small, we note that there has been extensive classical lattice-based cryptanalysis for factoring integers of this form [BDH99, CFRZ16] with other special constraints. Specifically, we can factor integers of the form $p^r q^s$ in polynomial time provided that $\max(r, s) \geq \mathsf{poly}(\log p)$. We do not believe these algorithms extend to our setting; roughly, these algorithms seem to be effective in factoring N when N has a prime factor p such that (a) N is divisible by a relatively large power of p; and (b) p is still not too small relative to N. In our setting where $N = p^2 q$ and q is very small, neither of these is the case. Regardless, we emphasize that there has been little classical cryptanalysis for factoring integers of the specific form we consider (beyond Mulder's aforementioned algorithm [Mul24]), and we leave this important direction to future work.

Finally, we remark that it is well-known that completely factoring an n-bit integer N in the special case where N is a prime power can be done in classical $\mathsf{poly}(n)$ time. This is because of two straightforward facts: (a) we can find integers $A, k > 1$ such that $N = A^k$ if they exist by computing $N^{1/k}$ for all possible values of k (and $k_{\max} < \log N$ since $N = A^k \geq 2^k$); and (b) we can efficiently test whether A is prime [AKS04].

Quantum Algorithms for Factoring. Shor's algorithm [Sho97] was the first to show that arbitrary n-bit integers could be factored using quantum circuits of size $\widetilde{O}(n^2)$. However, Shor's algorithm does not benefit if N has a small prime divisor or small squarefree part, like the classical algorithms by [Len87, Mul24] do. The same holds for Regev's [Reg25] improvement on Shor's algorithm to use $\widetilde{O}(n^{3/2})$ gates. A work by Ekerå and Håstad [EH17], later built upon by Chevignard et al. [CFS24], achieved a constant factor improvement in circuit costs that holds specifically for RSA integers (the product of two primes of roughly the same size), but the asymptotic scaling of the algorithms was unchanged.

To the best of our knowledge, the only polynomial-time[9] quantum factoring circuit that benefits by more than constant factors from special structure in N is the aforementioned Jacobi factoring circuit by [LPDS12], combined with the near-linear time algorithms for computing Jacobi symbols [Sch71, TY90, BS96, Möl08]. Putting these constructions together yields a circuit of only $\widetilde{O}(n)$ gates and space for finding a factor of N when N is not squarefree. Viewed in this context, one of our contributions is showing that we can further drive down the space and depth of this circuit when the squarefree part B of $N = A^2 B$ is much smaller than N.

[8] It may initially seem that this is subsumed by Lenstra's elliptic curve method [Len87]. However, the constant hidden in the big O is different between the two algorithms: for [Len87] it is $\sqrt{2}$, while for [Mul24] it is 1.

[9] An alternative approach to factoring with quantum computers is to use quantum subroutines (e.g. Grover search) inside classical factoring algorithms. The benefits this yields, which can include sublinear qubit count and gains from special structure in N, come at the expense of *superpolynomial* gate count and depth [BHLV17, BBM17, MBV20].

2.4 Sums of Phases

2.4.1 Basic Lemmas

Lemma 2.6. *For any $x \in \mathbb{R} \setminus \mathbb{Z}$ and positive integer M, we have:*

$$\sum_{k=0}^{M-1} \exp\left(-2\pi i k x\right) = \frac{1 - \exp(-2\pi i x M)}{1 - \exp(-2\pi i x)}.$$

Proof. This is just the summation formula for a geometric series. We require $x \notin \mathbb{Z}$ so that the denominator of the RHS is non-zero. □

Lemma 2.7. *For any $x \in \mathbb{R}$, we have $|1 - \exp(2\pi i x)| = 2 \cdot |\sin(\pi x)|$.*

Proof. We have:

$$\begin{aligned} |1 - \exp(2\pi i x)|^2 &= (1 - \cos(2\pi x))^2 + \sin(2\pi x)^2 \\ &= 2 - 2\cos(2\pi x) \\ &= 4\sin^2(\pi x). \end{aligned}$$

□

Corollary 2.8. *For any $x \in \mathbb{R}$, we have $|1 - \exp(2\pi i x)| \leq 2\pi |x|$.*

Proof. This is immediate from Lemma 2.7 and the well-known inequality that $|\sin x| \leq |x|$. □

Corollary 2.9. *For any $x \in \mathbb{R}$ such that $|x| \leq 1 - \Omega(1)$, we have $|1 - \exp(2\pi i x)| = \Omega(|x|)$.*

Proof. This is immediate from Lemma 2.7 and the fact that $|\sin x| = \Omega(|x|)$ for $x \in [-\pi + \Omega(1), \pi - \Omega(1)]$. □

We now combine these results in the following lemma:

Lemma 2.10. *For any $x \in \mathbb{R}$ and positive integer M such that $|xM| \leq 1 - \Omega(1)$, we have*

$$\left|\sum_{k=0}^{M-1} \exp(-2\pi i k x)\right| = \Theta(M).$$

Proof. First, if $x \in \mathbb{Z}$ then each term in the summation will be 1, so the LHS will be exactly M. Hence we assume from now on that $x \in \mathbb{R}/\mathbb{Z}$. In this case, the upper bound is straightforward: the LHS is $\leq M$ by a straightforward triangle inequality. For the lower bound, note that:

$$\begin{aligned} \left|\sum_{k=0}^{M-1} \exp(-2\pi i k x)\right| &= \left|\frac{1 - \exp(-2\pi i x M)}{1 - \exp(-2\pi i x)}\right| \text{ (Lemma 2.6)} \\ &= \Omega\left(\frac{|xM|)}{|1 - \exp(-2\pi i x)|}\right) \text{ (Corollary 2.9)} \\ &\geq \Omega(M) \text{ (Corollary 2.8).} \end{aligned}$$

□

2.4.2 Gauss Sums

Here, we state results that essentially imply that the Jacobi symbol is appropriately "pseudorandom" for the purposes of our algorithm and that of [LPDS12]. We follow the lecture notes by Conrad [Con].

Definition 2.11 (Dirichlet characters)**.** *For $m \in \mathbb{N}$, we say that $\chi : \mathbb{Z}_m \to \mathbb{C}$ is a* Dirichlet character mod m *if the following properties all hold:*

1. $\chi(a) = 0$ *if and only if* $\gcd(a, m) > 1$.
2. $\chi(ab) = \chi(a)\chi(b)$ *for all* a, b.

Definition 2.12 ((Im)primitive Dirichlet characters ([Con], Definition 3.3))**.** *We say that a Dirichlet character χ mod m is* imprimitive *if there is a proper divisor m' of m and a Dirichlet character χ' mod m' such that, for all $a \in \mathbb{Z}_m$ such that $\gcd(a, m) = 1$, we have $\chi(a) = \chi'(a \mod m')$.*

If χ is not imprimitive, we call it primitive.

Before continuing, we make a simple observation that the Jacobi symbol is a primitive Dirichlet character modulo any squarefree integer:

Lemma 2.13. *If $m > 1$ is odd and squarefree, then the Jacobi symbol $\chi(a) = \left(\frac{a}{m}\right)$ is a primitive Dirichlet character mod m.*

Proof. We know χ is a Dirichlet character from properties 1 and 2 of Theorem 2.4. It remains to check that it is primitive.

To this end, consider any proper divisor m' of m and a character χ' mod m'. Let $m = p_1 \dots p_r$ for distinct primes $p_1, \dots, p_r$ (since m is squarefree); since m' is a proper divisor of m, assume without loss of generality that p_1 does not divide m'.

Now consider $a \in \mathbb{Z}_m$ such that a is a quadratic non-residue mod p_1 and is congruent to 1 modulo $p_2, \dots, p_r$. Such a exists by the Chinese Remainder Theorem. Then $\chi(a) = -1$. On the other hand, since p_1 does not divide m' we have $a \bmod m' = 1 \Rightarrow \chi'(a \mod m') = \chi'(1) = 1$. (The final step is because we have $\chi'(1) = \chi'(1 \cdot 1) = \chi'(1)^2$ and $\chi'(1) \neq 0$, forcing $\chi'(1) = 1$.) Hence for this a, we have $\chi(a) \neq \chi'(a \bmod m')$. Such a exists for any m', χ', so χ is indeed primitive. □

Definition 2.14 (Gauss sums ([Con], Definition 3.1))**.** *For a Dirichlet character χ on $\mathbb{Z}_m$, we define its* Gauss sum *to be*

$$G(\chi) = \sum_{a \in \mathbb{Z}_m} \chi(a) \exp\left(\frac{2\pi i a}{m}\right).$$

Theorem 2.15 ([Con], Theorem 3.12)**.** *For any primitive Dirichlet character χ on $\mathbb{Z}_m$, we have $|G(\chi)| = \sqrt{m}$.*

This allows us to prove the specific form of the Gauss sum bound that we will need. We refer the reader to Section 1.1 for an overview of where these sums of phases come from, and reiterate the intuition here. Informally, if we want to recover Q given $N = P^2 Q$ as input, we will end up with a superposition of several periodic signals with period Q. The below lemma (with $m = Q$) examines the result of applying a QFT to this superposition, and tells us that these signals will essentially interfere like a randomly chosen collection of periodic signals.

Lemma 2.16. *Suppose m is odd and squarefree and consider any $k \in \mathbb{Z}_m$. Then we have:*

$$\left| \sum_{j \in \mathbb{Z}_m} \left(\frac{j}{m}\right) \exp\left(-\frac{2\pi i j k}{m}\right) \right| = \begin{cases} \sqrt{m}, & \text{if } \gcd(k,m) = 1, \\ 0, & \text{else.} \end{cases}.$$

Proof. First, we address the case where $\gcd(k,m) = 1$. Let χ denote the Jacobi symbol mod m. In this case we make a simple change of variables:

$$\begin{aligned} \sum_{j \in \mathbb{Z}_m} \left(\frac{j}{m}\right) \exp\left(-\frac{2\pi i j k}{m}\right) &= \sum_{j \in \mathbb{Z}_m} \left(\frac{-j/k}{m}\right) \exp\left(\frac{2\pi i j}{m}\right) \\ &= \left(\frac{-k}{m}\right) \cdot G(\chi), \end{aligned}$$

and now the conclusion is immediate from Theorem 2.15 and Lemma 2.13.

It remains to address the case where $\gcd(k,m) = d > 1$. Let $k' = k/d$ and $m' = m/d$. If $k \equiv 0 \pmod{m}$, the conclusion is immediate from Corollary 2.5, so we may assume $m' > 1$. Since m is squarefree, we have $\gcd(d, m') = 1$ and thus we can use the Chinese Remainder Theorem to associate a value $j \in \mathbb{Z}_m$ with its residues $j \bmod m'$ and $j \bmod d$. Bearing this in mind, we have:

$$\begin{aligned} \sum_{j \in \mathbb{Z}_m} \left(\frac{j}{m}\right) \exp\left(-\frac{2\pi i j k}{m}\right) &= \sum_{j \in \mathbb{Z}_m} \left(\frac{j}{m'}\right)\left(\frac{j}{d}\right) \exp\left(-\frac{2\pi i j k'}{m'}\right) \\ &= \sum_{j_1 \in \mathbb{Z}_d} \sum_{j_2 \in \mathbb{Z}_{m'}} \left(\frac{j_2}{m'}\right)\left(\frac{j_1}{d}\right) \exp\left(-\frac{2\pi i j_2 k'}{m'}\right) \\ &= \left(\sum_{j_1 \in \mathbb{Z}_d} \left(\frac{j_1}{d}\right)\right) \cdot \left(\sum_{j_2 \in \mathbb{Z}_{m'}} \left(\frac{j_2}{m'}\right) \exp\left(-\frac{2\pi i j_2 k'}{m'}\right)\right) \\ &= 0 \cdot \left(\sum_{j_2 \in \mathbb{Z}_{m'}} \left(\frac{j_2}{m'}\right) \exp\left(-\frac{2\pi i j_2 k'}{m'}\right)\right) \text{ (Corollary 2.5)} \\ &= 0, \end{aligned}$$

as desired. □

3 Factoring Squarefull Integers

In this section we prove the following theorem which refines the result by [LPDS12]. Crucially, we build on [LPDS12], showing that it suffices for the initial superposition to extend only to $\mathsf{poly}(B_{\max})$, rather than $\mathsf{poly}(N)$ (where $B_{\max}$ is an upper bound on B, explicitly defined in the theorem statement below). We will leverage this to factor a large class of integers with sublinear space and depth in Section 4.

Theorem 3.1. *Let N, n be positive integers such that $2^{n-1} \leq N < 2^n$, and let A, B be the unique positive integers such that B is squarefree and $N = A^2 B$. We will further assume that N is neither squarefree nor a square i.e. $A, B > 1$. Suppose there exists a quantum circuit that implements the operation*

$$|x\rangle |0^S\rangle \mapsto j_N(x) |x\rangle |0^S\rangle,$$

using $S := S(\ell, n)$ ancilla qubits with $G(\ell, n)$ gates and $D(\ell, n)$ depth, for any positive integer x such that $x < 2^\ell$. As in Definition 2.3, $j_N(x) \in \{-1, 1\}$ is such that $j_N(x) = \left(\frac{x}{N}\right)$ when $\gcd(x, N) = 1$ and can be arbitrary for other x.

Suppose we are also given an upper bound $B_{\max}$ on B, and define $\ell := \lfloor 2 \log B_{\max} \rfloor + \omega(1)$. Then there is a quantum algorithm that, given as input N and $B_{\max}$, outputs either B or a prime dividing N, with probability $\Omega(1)$. The quantum circuit uses

$$G(\ell, n) + O(\ell \log \ell)$$

gates, $D(\ell, n) + O(\ell)$ depth, and $S(\ell, n) + \ell$ qubits, and any classical pre/post-processing is polynomial-time.

Before we prove the theorem, notice that plugging in the Jacobi symbol algorithm due to [Sch71, TY90, BS96, Möl08] (discussed in Section 2.3) and simply setting $B_{\max} = N$ immediately yields the following corollary:[10]

Corollary 3.2 ([LPDS12, Sch71])**.** *Let N, A, B, n be as in Theorem 3.1. Then there exists a quantum algorithm that, given as input N, outputs either B or a prime dividing N, with probability $\Omega(1)$. The quantum circuit uses $\widetilde{O}(n)$ gates.*

We now turn to the proof of Theorem 3.1. Our algorithm is detailed in Algorithm 3.1 and very closely follows Shor's period-finding algorithm [Sho97]; the main difference is that we will end up with a superposition of multiple periodic signals with the same period rather than just one periodic signal. Nevertheless, we can argue using Gauss sums (see Section 2.4.2) that taking a QFT with this "somewhat periodic" signal still suffices to factor N.

We first address efficiency, then turn to correctness. We have the following costs:

- The uniform superposition over $[1, 2^\ell]$ can be initialized in depth 1 and gates ℓ, using no ancilla qubits.
- The Jacobi symbol computation can be carried out in depth $D(\ell, n)$, gates $G(\ell, n)$, and space $S(\ell, n)$ by supposition.
- For the QFT mod 2^ℓ, we rely on Coppersmith's $o(1)$-approximate QFT [Cop02], which uses $O(\ell \log \ell)$ gates, $O(\ell)$ depth, and no ancilla qubits.

As stated at the beginning of Algorithm 3.1, let $c > 1$ be a constant parameter. We will assume throughout this section that $N = A^2 B$, where B is squarefree, $A, B > 1$, and any prime divisor of A, B is $\geq n^c$. In our calculations, we will sometimes use big-O notation to denote an arbitrary *complex number* within a certain magnitude i.e. the notation $O(t)$ denotes some $z \in \mathbb{C}$ such that $|z| \leq O(t)$.

It now remains to prove the correctness of Algorithm 3.1. To do this, we first prove a preliminary technical lemma, then turn our attention to proving the theorem. The need for this technical lemma is twofold:

- We would like to argue that we can safely ignore inputs x in the superposition where $\gcd(x, N) > 1$, so we will have $j_N(x) \neq 0 = \left(\frac{x}{N}\right)$.

[10] We note that the original paper by [LPDS12] does not appear to state a result using Schönhage's near-linear size Jacobi algorithm [Sch71]; rather, they work with the better-known quadratic-time algorithms for computing the Jacobi symbol. Nevertheless, we credit [LPDS12] with this result since this result does essentially follow directly from their analysis (they need only set $\ell = \Theta(n)$ in Theorem 3.1).

Algorithm 3.1: The Jacobi Factoring Circuit for Squarefull Integers

Input: Positive integer $N = A^2B$ and a bound $B_{\max} \leq N$ such that $B \in [2, B_{\max}]$ is squarefree.

Output: Either the value of B, or a prime divisor p of N (with probability $\Omega(1)$).

1. First, we dispose of easy cases classically (in $\mathsf{poly}(n)$ time). Let $c > 1$ be some real constant parameter. If N has any prime divisor $\leq n^c$, output that prime and terminate. We may hence assume from now on that all prime divisors of A, B are $> n^c$, and hence also $B_{\max} \geq n^c$.

2. Set $\ell := \lfloor 2 \log B_{\max} \rfloor + 1$ and $S := S(\ell, n)$.

3. Initialize a uniform superposition
$$\frac{1}{2^{\ell/2}} \sum_{x=1}^{2^\ell} |x\rangle \, |0^S\rangle \, .$$

4. Compute the function $j_N(x)$ in superposition, to obtain the following state:
$$\frac{1}{2^{\ell/2}} \sum_{x=1}^{2^\ell} j_N(x) \, |x\rangle \, |0^S\rangle \, .$$

5. Apply a QFT mod 2^ℓ to the x register, and measure to obtain an integer $x^* \in [0, 2^\ell - 1]$.

6. Finally, for the classical post-processing, use the continued fraction expansion of $\frac{x^*}{2^\ell}$ (as in Shor [Sho97]; see [HW75, Chapter X] for details) to find positive integers X_1 and X_2 such that $X_2 \leq B_{\max}$ and $\left| \frac{x^*}{2^\ell} - \frac{X_1}{X_2} \right|$ is minimal. Output X_2 and terminate. (We will show that with probability $\Omega(1)$, we will in fact have $X_2 = B$.)

- Even when $\gcd(x, N) = 1$, we would ideally be able to say that for any x in our superposition, we have $\left(\frac{x}{N}\right) = \left(\frac{x}{A}\right)^2 \left(\frac{x}{B}\right) = \left(\frac{x}{B}\right)$, and hence after measuring we end up with a superposition over values x of $\left(\frac{x}{B}\right)|x\rangle$. The problem is that this is only true if $\left(\frac{x}{A}\right) \in \{-1, 1\}$. This is true most of the time, but there will be a small fraction of inputs x (specifically, those that share common factors with A but not B) such that $\left(\frac{x}{B}\right) \in \{-1, 1\}$ but $\left(\frac{x}{A}\right) = 0 \Rightarrow \left(\frac{x}{N}\right) = 0$.

The following lemma informally says that because there are not many x's where either of the above issues come up, we can safely ignore these technicalities: even though our algorithm prepares the state $|\psi_2\rangle$, we can safely pretend that it in fact prepares the simpler state $|\psi_1\rangle$ (by a trace distance argument).

Lemma 3.3. *Define $|\psi_1\rangle$ and $|\psi_2\rangle$ to be the following unnormalized states:*

$$|\psi_1\rangle = \sum_{1 \le x \le M} \left(\frac{x}{B}\right)|x\rangle$$
$$|\psi_2\rangle = \sum_{1 \le x \le M} j_N(x)\,|x\rangle\,.$$

Then the corresponding normalized states are $O(n^{(1-c)/4} \cdot (\log n)^{1/4})$-close in trace distance. Moreover, we have $\||\psi_1\rangle\|_2^2 = \frac{M\varphi(B)}{B}(1 + o(1))$. (Note that these two states may not be identical, for the aforementioned reasons.)

Proof. We will use [Che24, Lemma 2.11]. We first estimate $\||\psi_1\rangle\|_2$. We have:

$$\begin{aligned}
\||\psi_1\rangle\|_2^2 &= |\{x \in [1, M] \,:\, \gcd(x, B) = 1\}| \\
&= \sum_{j \in [1, B-1] \,:\, \gcd(x,B)=1} |\{x \in [1, M] \,:\, x \equiv j \bmod B\}| \\
&= \sum_{j \in [1, B-1] \,:\, \gcd(j,B)=1} \left(\left\lfloor \frac{M-j}{B} \right\rfloor - \left\lceil \frac{1-j}{B} \right\rceil + 1\right) \text{ (Proposition 2.1)} \\
&= \sum_{j \in [1, B-1] \,:\, \gcd(j,B)=1} \left(\frac{M-j}{B} - \frac{1-j}{B} + O(1)\right) \\
&= \frac{M-1}{B} \cdot \varphi(B) + O(\varphi(B)) \text{ (Corollary 2.5)} \\
&= \frac{M\varphi(B)}{B}(1 + o(1)) \text{ (noting that } M = \omega(B)).
\end{aligned}$$

Next, we upper bound $\||\psi_1\rangle - |\psi_2\rangle\|_2$. Let $p_1, \ldots, p_r$ be the distinct primes dividing N but not B. Note that r must be at most n, and moreover by assumption we have $p_i \ge n^c$ for all i. With this in mind, we have:

$$\begin{aligned}
\||\psi_1\rangle - |\psi_2\rangle\|_2^2 &\le O(1) \cdot |\{x \in [1, M] \,:\, \gcd(x, N) > 1\}| \\
&\le O(1) \cdot \sum_{i=1}^{r} |\{x \in [1, M] \,:\, p_i \mid x\}| \\
&\le O(1) \cdot \sum_{i=1}^{r} \frac{M}{p_i} \\
&\le O\left(\frac{M}{n^{c-1}}\right) \ (r \le n).
\end{aligned}$$

It then follows by [Che24, Lemma 2.11] that the trace distance we are concerned with is at most:

$$\begin{aligned} O\left(\sqrt{\frac{\||\psi_1\rangle - |\psi_2\rangle\|_2}{\||\psi_1\rangle\|_2}}\right) &\leq O\left(\sqrt[4]{\frac{M/n^{c-1}}{M\varphi(B)/B}}\right) \\ &\leq O\left(n^{(1-c)/4} \cdot \left(\frac{B}{\varphi(B)}\right)^{1/4}\right) \\ &\leq O\left(n^{(1-c)/4} \cdot (\log\log B)^{1/4}\right) \\ &\leq O\left(n^{(1-c)/4} \cdot (\log n)^{1/4}\right), \end{aligned}$$

as desired. □

We now prove Theorem 3.1. Our proof breaks down into a few steps: we will first set up some notation and write out the state computed by the algorithm after the QFT. We then lower bound the amplitude this state places on certain values $y \in [0, 2^\ell - 1]$, and use this to complete the proof.

Step 1: notation and setup. After computing the function $j_N(x)$, we have the state

$$\frac{1}{2^{\ell/2}} \sum_{x=1}^{2^\ell} j_N(x) |x\rangle |0^S\rangle .$$

We can now use Lemma 3.3 to change this to the state

$$|\psi_1\rangle = \sqrt{\frac{(1+o(1))B}{2^\ell \varphi(B)}} \sum_{1 \leq x \leq 2^\ell} \left(\frac{x}{B}\right) |x\rangle ,$$

incurring a trace distance loss of only $O(n^{(1-c)/4}) = o(1)$. The normalization factor follows from Lemma 3.3. After the QFT, we obtain the state:

$$\sqrt{\frac{(1+o(1))B}{2^{2\ell}\varphi(B)}} \sum_{y=0}^{2^\ell - 1} \left(\sum_{1 \leq x \leq 2^\ell} \left(\frac{x}{B}\right) \exp\left(-\frac{2\pi i x y}{2^\ell}\right)\right) |y\rangle . \tag{2}$$

At a high level, our analysis from this point mirrors the analysis by Shor [Sho97] of his period-finding procedure; we would like to show that this state places $\Omega(1)$ weight on states $|y\rangle$ such that $y/2^\ell$ is close to a multiple of $1/B$ with numerator relatively prime to B.

Step 2: lower bounding the amplitude on $|y\rangle$. In this section, we will lower bound the magnitude of the amplitude on $|y\rangle$ in Equation (2). Let $M = 2^\ell$ and $\epsilon = \frac{1}{2M}$ (this will be our target closeness bound). Note then by definition of ℓ (in Algorithm 3.1) that $M > B_{\max}^2$. It will be convenient for us to write $M = qB + r$, where $0 < r < B$ (we can assume M is not divisible by B since $B > 1$ is odd).

Lemma 3.4. *Consider a fixed $y \in [0, M-1]$ such that there exists an integer $k \in [1, B-1]$ and $\delta \in [-\epsilon, \epsilon]$ such that $\gcd(k, B) = 1$ and $\frac{y}{M} = \frac{k}{B} + \delta$. (Note in particular that this means $y \neq 0$.) Then we have:*

$$\left| \sum_{1 \leq x \leq M} \left(\frac{x}{B}\right) \exp\left(-\frac{2\pi i x y}{M}\right)\right| \geq \Omega(q\sqrt{B}).$$

Proof. The high-level idea is to use the fact that $\frac{y}{M} \approx \frac{k}{B}$ to replace $\frac{y}{M}$ in the LHS with $\frac{k}{B}$. This will of course not be completely correct, but we will carefully track the errors that arise from doing this. This will allow us to obtain the desired lower bound using a Gauss sum modulo B (see Lemma 2.16). We proceed as follows:

$$\begin{aligned}
\sum_{1\le x\le M}\left(\frac{x}{B}\right)\exp\left(-\frac{2\pi ixy}{M}\right) &= \sum_{1\le j\le B-1}\sum_{\substack{1\le x\le M\\ x\equiv j \bmod B}}\left(\frac{j}{B}\right)\exp\left(-\frac{2\pi ixy}{M}\right)\\
&= \sum_{1\le j\le B-1}\sum_{0\le l\le\left\lfloor\frac{M-j}{B}\right\rfloor}\left(\frac{j}{B}\right)\exp\left(-\frac{2\pi i(lB+j)y}{M}\right) \text{ (writing } x = lB+j)\\
&= \sum_{j\in[1,B-1]}\left[\left(\frac{j}{B}\right)\exp\left(-\frac{2\pi ijy}{M}\right)\cdot\sum_{0\le l\le\left\lfloor\frac{M-j}{B}\right\rfloor}\exp\left(-\frac{2\pi ilBy}{M}\right)\right]. \qquad (3)
\end{aligned}$$

We now analyze the inner sum. Note that $M-j = qB+r-j \Rightarrow \left\lfloor\frac{M-j}{B}\right\rfloor \in \{q-1, q\}$. We hence have:

$$\begin{aligned}
\sum_{l=0}^{\left\lfloor\frac{M-j}{B}\right\rfloor}\exp\left(-\frac{2\pi ilBy}{M}\right) &= \sum_{l=0}^{q-1}\exp\left(-\frac{2\pi ilBy}{M}\right)+O(1)\\
&= \sum_{l=0}^{q-1}\exp\left(-2\pi ilB\left(\frac{k}{B}+\delta\right)\right)+O(1)\\
&= \sum_{l=0}^{q-1}\exp\left(-2\pi ilB\delta\right)+O(1)\\
&= R+O(1),
\end{aligned}$$

where we define $R := \sum_{l=0}^{q-1}\exp(-2\pi ilB\delta)$. Since $|qB\delta| \le qB\epsilon \le 1/2$, we have by Lemma 2.10 that $|R| = \Theta(q)$. Bearing this in mind, we plug this into and continue from Equation (3) as follows:

$$\begin{aligned}
&\sum_{j\in[1,B-1]}\left[\left(\frac{j}{B}\right)\exp\left(-\frac{2\pi ijy}{M}\right)\cdot\sum_{0\le l\le\left\lfloor\frac{M-j}{B}\right\rfloor}\exp\left(-\frac{2\pi ilBy}{M}\right)\right]\\
=&\sum_{j\in[1,B-1]}\left[\left(\frac{j}{B}\right)\exp\left(-\frac{2\pi ijy}{M}\right)\cdot(R+O(1))\right]\\
=&R\cdot\left[\sum_{j\in[1,B-1]}\left(\frac{j}{B}\right)\exp\left(-\frac{2\pi ijy}{M}\right)\right]+O(\varphi(B))\\
=&R\cdot\left[\sum_{j\in[1,B-1]}\left(\frac{j}{B}\right)\exp\left(-2\pi ij\left(\frac{k}{B}+\delta\right)\right)\right]+O(\varphi(B))\\
=&R\cdot\left[\sum_{j\in[1,B-1]}\left(\frac{j}{B}\right)\exp\left(-\frac{2\pi ijk}{B}\right)(1+O(B\epsilon))\right]+O(\varphi(B)) \text{ (Corollary 2.8)}\\
=&R\cdot\left[\left(\sum_{j\in[1,B-1]}\left(\frac{j}{B}\right)\exp\left(-\frac{2\pi ijk}{B}\right)\right)+O(B\varphi(B)\epsilon)\right]+O(\varphi(B))
\end{aligned}$$

$$= R \cdot \left[\sqrt{B} + O(1) \right] + O(\varphi(B)) \text{ (Lemma 2.16).}$$

The final step follows from Gauss sums; we state their key properties (including Lemma 2.16) in Section 2.4.2. We also use the fact that $B\varphi(B)\epsilon \leq B^2/M \leq 1$. Finally, we lower bound the magnitude of this amplitude as follows:

$$\begin{aligned}
&\left| R \cdot \left[\sqrt{B} + O(1) \right] + O(\varphi(B)) \right| \\
\geq &\left| R \cdot \sqrt{B} \right| - O\left(|R|\right) - O(\varphi(B)) \\
\geq &\Omega(q\sqrt{B}) - O(q) - O(\varphi(B)) \text{ (since } |R| = \Theta(q)) \\
\geq &\Omega(q\sqrt{B}),
\end{aligned}$$

as desired. To justify the final step, note that $q\sqrt{B} \geq q\sqrt{n^c} \gg q \geq B \geq \varphi(B)$. □

Proof of Theorem 3.1. Call $y \in [0, M-1]$ *successful* if there exists an integer $k \in [1, B-1]$ and real $\delta \in [-\epsilon, \epsilon]$ such that $\gcd(k, B) = 1$ and $\frac{y}{M} = \frac{k}{B} + \delta$. By Lemma 3.4 and Equation (2), for any successful y, the probability that we measure and get the classical outcome $x^* = y$ is at least

$$\Omega\left(\frac{B}{M^2\varphi(B)} \cdot (q\sqrt{B})^2 \right) = \Omega\left(\frac{q^2B^2}{M^2\varphi(B)} \right) \geq \Omega\left(\frac{1}{\varphi(B)} \right).$$

Next, we argue that there are at least $\varphi(B)$ successful values of y. Indeed, for any $k \in [1, B-1]$ such that $\gcd(k, B) = 1$, consider the interval $\left[\frac{k}{B} - \epsilon, \frac{k}{B} + \epsilon \right] \subset (0, 1)$. It has width $2\epsilon = \frac{1}{M}$, so there must be at least one multiple of $1/M$ in this interval i.e. there exists an integer y_k such that $|\frac{y_k}{M} - \frac{k}{B}| \leq \epsilon$ (which in turn implies $y_k/M \in (0, 1) \Rightarrow y_k \in [1, M-1]$); in other words, y_k is successful. Moreover, we claim that $y_k \neq y_{k'}$ for any $k \neq k'$. If this were not true, then the triangle inequality would force $\left| \frac{k}{B} - \frac{k'}{B} \right| \leq 2\epsilon \Rightarrow \frac{1}{B} \leq 2\epsilon$, which is false. Since there are $\varphi(B)$ many such values of k, there are at least $\varphi(B)$ distinct successful values of y, each of which we obtain with probability $\geq \Omega(\frac{1}{\varphi(B)})$. It follows that with $\Omega(1)$ probability, we will obtain such a y, as claimed.

Now to finish, we have some integer $y = x^*$ such that $\left| \frac{y}{M} - \frac{k}{B} \right| \leq \frac{1}{2M} < \frac{1}{2B_{\max}^2}$. It follows that k/B is the closest fraction to $\frac{y}{M}$ with denominator at most $B_{\max}$. Hence our algorithm will obtain $X_1 = k$ and $X_2 = B$ in the final step, and output the denominator B. This completes the proof of the theorem. □

Remark 1. *Algorithm 3.1, and its associated Theorem 3.1, receive as input a bound $B_{\max}$ on the size of B, the squarefree part of the input integer N. Here we note that if $B_{\max}$ is not known, the algorithm can still be used to find B (or a prime dividing N) with high probability, and with roughly the same quantum circuit sizes, as follows. Via Lemma 5.2, for any $B_{\max} > B$ the probability of success of the algorithm can be boosted to $1 - \epsilon$ using $O(\log 1/\epsilon)$ calls to Algorithm 3.1 and a small amount of classical computation. Starting with some $B_{\max} = O(1)$, this larger algorithm can then be iterated, doubling $\log B_{\max}$ every iteration until a value that divides N is found and the algorithm halts. The number of iterations is expected to be $\log \log B$ and with high probability the algorithm will halt with $\log B_{\max} < 2 \log B$ on the last iteration. This implies that the complexity of the algorithm will be only worse by a constant factor if $B_{\max}$ is not supplied.*

Remark 2. *In Algorithm 3.1, we take a superposition over all $x < O(B_{\max}^2)$ in order to recover the period of a periodic function with period $\leq B_{\max}$. This is in direct analogy with Shor's original period-finding*

subroutine [Sho97]: to factor an integer N, Shor considers a periodic function with period $\leq N$ and takes a superposition over all inputs $x < O(N^2)$ to recover this period.

Works subsequent to Shor's original paper [Sei01, EH17] show that it can suffice to use a superposition only over all $x < O(N^{1+\epsilon})$ for any $\epsilon > 0$. The circuit must then be run independently $O(1/\epsilon)$ times; the period is subsequently recovered using a more sophisticated classical post-processing procedure. Analogously, we believe it is likely possible to modify Algorithm 3.1 to only take the superposition up to $O(B_{\max}^{1+\epsilon})$, and run the resulting circuit $O(1/\epsilon)$ times and classically post-process the results as in [Sei01]. This would enable constant-factor improvements to the space and depth, which would be important when instantiating this circuit in practice as a proof of quantumness (see Corollary 4.8).

Remark 3 (On the use of Gauss sums)**.** *In this analysis, we made use of the "strong half" of Lemma 2.16, namely its conclusion in the case where* $\gcd(k, m) = 1$*. On the other hand, the original analysis by [LPDS12] only makes use of the "weak half" i.e. the case where* $\gcd(k, m) > 1$*. The reason we use Gauss sums in all their power is because of our adaptation of Shor's analysis [Sho97] to the setting where the initial superposition ranges only up to* $\mathsf{poly}(Q)$*. As in Shor's analysis, we lower bounded the amplitude of the final (post-QFT) state on each "useful" value and this requires a strong Gauss sum bound.*

If we wanted to get around the need for the strong Gauss sum bound, we could instead adapt the tighter analysis of Regev's factoring algorithm [Reg25] based on the Poisson summation formula. This would only require the "weak" Gauss sum result (as in [LPDS12]), but would likely require us to start with a discrete Gaussian superposition instead of a uniform superposition. Given the concrete overheads involved in preparing discrete Gaussian superpositions [GR02, Reg09] and our interest in the potential practicality of our algorithms, we chose to present the simpler algorithm with just a uniform superposition.

We thank Oded Regev for pointing out to us that the use of "strong" Gauss sum bounds in our analysis could be circumvented.

4 Algorithm for Computing Jacobi Symbols

In this section we present one of our core technical contributions: an algorithm to compute the Jacobi symbol of x mod N, where N is classical and x could be in superposition. We remark that our algorithm is also readily adaptable to computing the gcd of x and N, much like other algorithms for computing the Jacobi symbol [Sch71, TY90, BS96, Möl08]. When $N < 2^n$ and $x < 2^m$, our construction requires circuit-size $\widetilde{O}(n)$ and space $\widetilde{O}(m)$, which we can exploit due to our analysis in Section 3 which allows us to restrict $m \ll n$. In contrast, the 2012 result of Li, Peng, Du and Suter [LPDS12], together with near-linear time algorithms due to [Sch71, TY90, BS96, Möl08] for computing Jacobi symbols, uses gates *and space* $\widetilde{O}(n)$ (see Corollary 3.2 for a formal statement of this result by [LPDS12]). Our improvements are thus along the axes of space and, as we will see in Section 4.2, depth.

We begin with an abstract algorithm (formalized in Theorem 4.1) that makes black-box use of circuits for multiplying and for computing the Jacobi symbol between equally-sized inputs. We then instantiate the circuits using explicit constructions for these subroutines [Sch71, TY90, BS96, Möl08, NZLS23] in Section 4.2.

4.1 Abstract Construction

Let us first summarize the main idea of our construction; we refer the reader to the technical overview in Section 1.2 for further discussion of our high-level approach.

The standard algorithms for computing the Jacobi symbol are the extended Euclidean algorithm and the binary GCD; out of the box, neither one achieves the efficiency we desire, in particular when one input is much smaller than the other. Indeed, existing circuits for both algorithms require space proportional to the length of the larger input. Nevertheless, we can draw inspiration from an observation about the extended Euclidean algorithm: after just one iteration, the larger input is reduced to roughly the same size as the smaller one, and the entire rest of the computation has cost that scales only with the length of the smaller input. Our task thus boils down to "just" computing N mod x in sublinear space. The naive idea would be to use standard long division or a variant thereof. However, while long division only needs to look at $O(\log(x))$ bits of N at a time, we have to keep track of the intermediate values to make the computation reversible. Sadly, keeping track of these excess values appears to require linear space.

Our solution, which is inspired by the binary GCD algorithm [BS96] and strongly resembles Montgomery reduction [Mon85], is to perform a "flipped" variant of long division. In the Euclidean algorithm, the computation of N mod x can be thought of as a way to find a multiple κx such that $N - \kappa x < x$ — in particular, all but the lowest m bits of $N - \kappa x$ are zero. Long division achieves this by starting from the most significant bit (MSB) and iterating towards the least significant bit (LSB), zeroing out bits along the way. We flip this idea on its head: we compute a multiple kx such that all but the *highest-order* m bits of $N - kx$ are zero—by starting from the LSB and iterating towards the MSB. From here, there are three key ideas:

1. The value kx can be built up bit by bit via a loop in which each iteration is entirely reversible. We can easily uncompute intermediate states by performing a simple comparison that depends only on the MSBs of our current state.

2. Furthermore, once a lower-order bit of kx matches that of N, this bit will not depend on anything quantum (since N is classical). This qubit is thus in a classically-known state, and unentangled from all other qubits in the computation. We can use this fact to recycle lower-order qubits as we set them. Ultimately, we only ever store the leading $O(m)$ bits of the partial value $y = kx$ that is being computed. In Algorithm 4.2, we will denote the register holding this sliding window of $O(m)$ bits as z.

3. The above two ideas on their own already suffice to obtain a circuit that computes the Jacobi symbol in sublinear space $O(m)$. However, its gate count will be $O(nm)$. This is because, when setting one bit of k at a time, we will ultimately have to carry out "schoolbook" arithmetic comprising additions.

   Instead, we can set k in *batches* of m bits each. This allows us to benefit from fast integer multiplication algorithms [SS71, HvdH21, KMY24], and will ultimately drop the gate count to $\widetilde{O}(n)$, which is near-linear.

At the end, our quantum computer will hold the value $(N - kx)/2^{n-m}$, where kx is a multiple such that both of the following are true (where we let $m \geq \lceil \log x \rceil$): (a) $N - kx$ is divisible by 2^{n-m}; and (b) $kx < 2^n$. We can now proceed from here in one of two ways:

- This actually suffices to complete the first step of extended Euclidean and compute N mod x, since we have:

$$N \equiv \frac{N - kx}{2^{n-m}} \cdot 2^{n-m} \pmod{x},$$

and $2^{n-m} \bmod x$ is very efficiently computable. This is asymptotically a satisfactory solution, but adds unnecessary indirection and concrete efficiency to our overall algorithm, as we will see next.

We thank Daniel J. Bernstein for pointing out this variant of our algorithm to us.

- Rather than taking the extra step as above to compute $N \bmod x$, we could just start from $(N-kx)/2^{n-m}$ and directly carry out the following chain of transformations to compute $\left(\frac{x}{N}\right)$, using the properties of the Jacobi symbol stated in Theorem 2.4:

$$\left(\frac{x}{N}\right) \to \left(\frac{N}{x}\right) \to \left(\frac{N-kx}{x}\right) \to \left(\frac{(N-kx)/2^{n-m}}{x}\right).$$

This is the approach that we will take, as detailed in Algorithm 4.1.

We first describe how to utilize the value kx to compute the Jacobi symbol (Algorithm 4.1), then present our procedure for obtaining the value kx (Algorithm 4.2), and then finally prove our claimed performance guarantees.

Algorithm 4.1: Reversible algorithm for computing Jacobi symbols

---

Data: Efficiency parameters m, n such that $m|n$, and positive integers $N < 2^n$ and $x < 2^m$

Result: The Jacobi symbol $\left(\frac{x}{N}\right)$

1. Compute the integers x' and t such that $x' = x/2^t$ is an odd integer.
2. Set the register out, which will ultimately store the Jacobi symbol $\left(\frac{x}{N}\right)$, as follows:

$$\mathsf{out} = \left((-1)^{\frac{N^2-1}{8}}\right)^t \cdot (-1)^{\frac{(x'-1)(N-1)}{4}} \cdot \left((-1)^{\frac{x'^2-1}{8}}\right)^{n-m}. \tag{4}$$

3. Use Algorithm 4.2 on inputs N and x' to compute some integer z, then set $s = \left\lfloor \frac{N}{2^{n-m}} \right\rfloor - z$. (We will show in Lemma 4.4 that $s = (N - kx')/2^{n-m}$ for some integer k. Note that s could be negative.)
4. Compute $\mathsf{out} \leftarrow \mathsf{out} \cdot \left(\frac{s}{x'}\right)$, where the Jacobi symbol $\left(\frac{s}{x'}\right)$ is computed via the algorithms of [Sch71, TY90, BS96, Möl08], made reversible via standard techniques [Ben73, Ben89, LS90].
5. Uncompute s, z, x', and t by running steps 3 and 1 in reverse.
6. Return out.

Theorem 4.1. *Suppose there exists a quantum multiplication circuit on t-bit inputs with gates $G_{\text{mult}}(t)$, space $S_{\text{mult}}(t)$, and depth $D_{\text{mult}}(t)$. Also, suppose there exists a quantum circuit for computing the Jacobi symbol between two t-bit inputs with gates $G_{\text{Jac}}(t)$, space $S_{\text{Jac}}(t)$, and depth $D_{\text{Jac}}(t)$.*

Let $N < 2^n$ be a classically known odd integer. Then, there exists a quantum circuit implementing the unitary

$$|x\rangle |0\rangle^{\otimes 2} \mapsto |x\rangle \left|\left(\frac{x}{N}\right)\right\rangle, \tag{5}$$

acting on m-qubit quantum inputs $x \in [0, 2^m - 1]$ that runs in gates $O\left(\frac{n}{m} \cdot G_{\text{mult}}(m) + G_{\text{Jac}}(m) + m \log m\right)$, space $O\left(\max(S_{\text{mult}}(m), S_{\text{Jac}}(m))\right)$, and depth $O\left((\frac{n}{m} + \log m) \cdot D_{\text{mult}}(m) + D_{\text{Jac}}(m) + \log^2 m\right)$.

Algorithm 4.2: Reversible subroutine for finding a value kx, whose $n-m$ lowest-order bits equal the corresponding bits of N

Data: Efficiency parameters m, n such that $m|n$, and positive integers $N < 2^n$ and $x < 2^m$ with x odd

Result: The highest-order m bits of $y = kx$ for some k, such that $kx < 2^n$ and $N - kx$ is divisible by 2^{n-m}.

1. Set a $2m$ bit register $z = 0$. The low-order half of this register will ultimately store the leading m bits of y. (All other, lower-order bits of y match the corresponding bits of N, and thus do not need to be stored explicitly.)

2. Precompute the following values:

   (a) x_{minv}, the inverse of x mod 2^m

   (b) $x_{\text{inv}} = \frac{1}{x}$ with $2m$ bits of precision

3. Repeat the following for $j \in \{0, 1, 2, \ldots \frac{n-2m}{m}\}$

   (a) Compute an m-bit register $\mathsf{ctrl} = \left[x_{\text{minv}} \cdot (N_j - z)\right] \bmod 2^m$, where $N_j = \lfloor N/2^{jm} \rfloor \bmod 2^m$.

   (b) Compute $z \leftarrow z + \mathsf{ctrl} \cdot x$. Now, $z \bmod 2^m = N_j$.

   (c) Using z and x_{inv}, uncompute $\mathsf{ctrl} \leftarrow \mathsf{ctrl} \oplus \lfloor \frac{z}{x} \rfloor$ via [RV24, Lemma A.2].

   (d) Zero the m lowest-order bits of z using N_j, then swap them with the highest-order m bits: $z \leftarrow \lfloor \frac{z}{2^m} \rfloor$.

4. Uncompute the values from Step 2.

5. Return z.

Remark 4. *We state our result in terms of explicitly writing down the Jacobi symbol for the purposes of clarity and to emphasize that this also works equally well in the classical reversible setting. However, as detailed in Section 3, our algorithms will ultimately apply the Jacobi symbol in the* phase*, thus computing the unitary* $|x\rangle \mapsto j_N(x)\,|x\rangle$ *for some function* $j_N(x)$ *that is equal to the Jacobi symbol whenever* $\gcd(x, N) = 1$*. This can easily be achieved either by using the above algorithm out of the box and applying a phase kickback [CEMM98], or by going through Algorithms 4.1 and 4.2 and modifying them to write the output in the phase rather than in a single-qubit register.*

Correctness. We first show the correctness of Algorithm 4.2. For each value of the iteration index $j = 0, 1, \ldots, \frac{n-2m}{m}$ for the loop in step 3, we define the following variables:

- z_j: the value stored in register z at the beginning of iteration j of the loop;
- ctrl_j: the value of ctrl computed in iteration j;
- z'_j: the intermediate value of z in iteration j i.e. $z_j + \mathsf{ctrl}_j \cdot x$; and
- y_j: this is defined as $z_j \cdot 2^{jm} + (N \bmod 2^{jm})$. This is the multiple of x that we are tracking implicitly throughout the algorithm; we use y_j to represent the value of this multiple at the beginning of iteration j.

Also note the definition of N_j in step 3 of Algorithm 4.2, and let $z_{(n-m)/m}$ denote the value stored in register z at the end of the algorithm i.e. the final output. As we will see, correctness of Algorithm 4.2 boils down to the following lemma:

Lemma 4.2. *For all* $j = 0, 1, \ldots, (n-m)/m$*, all of the following hold:*

1. $y_j \equiv N \pmod{2^{jm}}$*;*
2. $0 \le y_j < 2^{jm} \cdot x$*; and*
3. y_j *is divisible by* x*.*

Proof. Item 1 is straightforward. To establish the other two items, we proceed by induction on j. For the base case where $j = 0$, we have $z_0 = 0 \Rightarrow y_0 = 0$. All three conditions are now evident. Now, for the inductive step, assume that we have shown the result for some $j \ge 0$ and wish to show the result for $j + 1$. We will examine the execution of iteration j of step 3 of Algorithm 4.2 to complete the induction. Firstly, by definition we have the following:

$$\begin{aligned} z'_j &= z_j + \mathsf{ctrl}_j \cdot x \\ &\equiv z_j + x_{\text{minv}} \cdot (N_j - z_j) \cdot x \pmod{2^m} \\ &\equiv N_j \pmod{2^m} \\ \Rightarrow 2^{jm} z'_j &\equiv 2^{jm} N_j \pmod{2^{(j+1)m}} \\ \Rightarrow 2^{jm} z'_j + \left(N \bmod 2^{jm}\right) &\equiv 2^{jm} N_j + \left(N \bmod 2^{jm}\right) \pmod{2^{(j+1)m}} \\ &\equiv N \pmod{2^{(j+1)m}}. \end{aligned} \tag{6}$$

This implies that the m lowest-order bits of z'_j match N_j and thus we can indeed use the bits of N_j to zero out the m lowest-order bits of z'_j. Hence we have

$$2^{jm} z'_j + (N \bmod 2^{jm}) = 2^{(j+1)m} z_{j+1} + (N \bmod 2^{(j+1)m}),$$

Now, we also have that:

$$\begin{aligned} y_{j+1} &= 2^{(j+1)m} z_{j+1} + (N \bmod 2^{(j+1)m}) \\ &= 2^{jm} z'_j + (N \bmod 2^{jm}) \\ &= 2^{jm}(z_j + \mathsf{ctrl}_j \cdot x) + (N \bmod 2^{jm}) \\ &= y_j + 2^{jm} \cdot \mathsf{ctrl}_j \cdot x. \end{aligned} \tag{7}$$

Since y_j is divisible by x by the induction hypothesis, this immediately implies condition 3. Finally, we can obtain condition 2 since:

$$\begin{aligned} y_{j+1} &= y_j + 2^{jm} \cdot \mathsf{ctrl}_j \cdot x \\ &< 2^{jm} \cdot x + 2^{jm} \cdot \mathsf{ctrl}_j \cdot x \text{ (induction hypothesis)} \\ &\leq 2^{jm} \cdot x + 2^{jm} \cdot (2^m - 1) \cdot x \text{ (since } \mathsf{ctrl}_j \text{ is reduced mod } 2^m) \\ &= 2^{(j+1)m} x. \end{aligned}$$

□

We complete our proof of correctness for Algorithm 4.2 with the following claim:

Proposition 4.3. *For all $j = 0, 1, \ldots, (n-m)/m$, we have $0 \leq z_j < x$. For $j = 0, 1, \ldots, (n-2m)/m$, we have $0 \leq z'_j < 2^{2m}$. Moreover, we have $\mathsf{ctrl}_j = \left\lfloor \frac{z'_j}{x} \right\rfloor$. (This establishes that $2m$ qubits are sufficient to hold the z register, and that the uncomputation of ctrl_j proceeds correctly.)*

Proof. Since $y_j = z_j \cdot 2^{jm} + (N \bmod 2^{jm})$, we have:

$$\begin{aligned} z_j &= \left\lfloor \frac{y_j}{2^{jm}} \right\rfloor \\ &\leq \left\lfloor \frac{(2^{jm} - 1)x}{2^{jm}} \right\rfloor \text{ (Lemma 4.2)} \\ &\in [0, x-1]. \end{aligned}$$

Since $z'_j = z_j + \mathsf{ctrl}_j \cdot x$, it follows that $\mathsf{ctrl}_j = \left\lfloor \frac{z'_j}{x} \right\rfloor$. Finally, we have:

$$\begin{aligned} z'_j &= z_j + \mathsf{ctrl}_j \cdot x \\ &< (1 + \mathsf{ctrl}_j) \cdot x \\ &\leq 2^m \cdot x \text{ (}\mathsf{ctrl}_j \text{ is reduced mod } 2^m\text{)} \\ &< 2^{2m}, \end{aligned}$$

as desired. □

Finally, we show the correctness of Algorithm 4.1:

Lemma 4.4. *Algorithm 4.1 correctly computes the Jacobi symbol* $\left(\frac{x}{N}\right)$. *Moreover, we have* $|s| < 2^m$ *(thus the step of computing* $\left(\frac{s}{x'}\right)$ *only needs to work with m-bit inputs).*

Proof. We retain all notation introduced in Algorithm 4.1. We first show that $|s| < 2^m$ and that there exists an integer k such that $N - kx' = 2^{n-m} \cdot s$. To this end, recall that the output of Algorithm 4.2 is exactly $z = z_{(n-m)/m}$, where $z_{(n-m)/m} \cdot 2^{n-m} + (N \bmod 2^{n-m}) = y_{(n-m)/m}$ is equal to kx' for some integer k by Lemma 4.2. Then note firstly that:

$$\begin{aligned} |s| &\leq \max\left(\left\lfloor \frac{N}{2^{n-m}} \right\rfloor, z_{(n-m)/m}\right) \\ &< 2^m, \end{aligned}$$

since $N < 2^n$ and $z_{(n-m)/m} < x \leq 2^m$ by Proposition 4.3. Secondly, we have:

$$\begin{aligned} 2^{n-m} \cdot s &= 2^{n-m} \cdot \left(\left\lfloor \frac{N}{2^{n-m}} \right\rfloor - z_{(n-m)/m}\right) \cdot \\ &= 2^{n-m} \cdot \left\lfloor \frac{N}{2^{n-m}} \right\rfloor - 2^{n-m} \cdot z_{(n-m)/m} \\ &= N - (N \bmod 2^{n-m}) - 2^{n-m} \cdot z_{(n-m)/m} \\ &= N - y_{(n-m)/m} \\ &= N - kx'. \end{aligned}$$

The conclusion will now follow from the standard properties of the Jacobi symbol stated in Theorem 2.4:

$$\begin{aligned} \left(\frac{x}{N}\right) &= \left(\frac{2^t x'}{N}\right) \\ &= \left((-1)^{\frac{N^2-1}{8}}\right)^t \cdot \left(\frac{x'}{N}\right) \text{ (Theorem 2.4, properties 2 and 6)} \\ &= \left((-1)^{\frac{N^2-1}{8}}\right)^t \cdot (-1)^{\frac{(x'-1)(N-1)}{4}} \cdot \left(\frac{N}{x'}\right) \text{ (Theorem 2.4, property 7)} \\ &= \left((-1)^{\frac{N^2-1}{8}}\right)^t \cdot (-1)^{\frac{(x'-1)(N-1)}{4}} \cdot \left(\frac{N - kx'}{x'}\right) \text{ (Theorem 2.4, property 4)} \\ &= \left((-1)^{\frac{N^2-1}{8}}\right)^t \cdot (-1)^{\frac{(x'-1)(N-1)}{4}} \cdot \left(\frac{2^{n-m} \cdot s}{x'}\right) \\ &= \left((-1)^{\frac{N^2-1}{8}}\right)^t \cdot (-1)^{\frac{(x'-1)(N-1)}{4}} \cdot \left((-1)^{\frac{x'^2-1}{8}}\right)^{n-m} \cdot \left(\frac{s}{x'}\right), \text{ (Theorem 2.4, properties 2 and 6)} \end{aligned}$$

which implies the conclusion. □

Efficiency. We now turn our attention to establishing the desired efficiency guarantees:

Lemma 4.5. *Algorithm 4.1 achieves the efficiency guarantees claimed in Theorem 4.1.*

Proof. We proceed by showing that each step of Algorithm 4.1 can be implemented reversibly with the stated complexities. We note that $S_{\text{Jac}}(m)$ must be at least linear in m because the Jacobi symbol depends on the entire input; therefore our space complexity is lower bounded by $\Omega(m)$.

Step 1 computes the number of trailing zeros t of a length-m value x, and then computes x', which is x shifted to the right by t bits. This can be performed reversibly in $O(m \log m)$ gates, $O(m)$ space, and $O(\log^2 m)$ depth, as follows. We first compute t by using a tree of $O(m)$ Toffoli gates to compute the unary representations of $\lfloor t/2^i \rfloor$ for i from 1 up to $\lceil \log m \rceil$, and then a tree of controlled-NOT gates to compute the parity of each unary value, which is equal to bit i of t. The value x' can then be computed by applying the map $|a\rangle \to |\lfloor a/2^{it_i} \rfloor\rangle$ repeatedly for each bit t_i of t, beginning with $a = x$. In turn, this map can be implemented with m ancilla bits by applying the out-of-place controlled bit-shift map $|a\rangle|0\rangle \to |a\rangle|\lfloor a/2^{it_i} \rfloor\rangle$, followed by $|a\rangle|b\rangle \to |a \oplus 2^{it_i} b\rangle|b\rangle$ to uncompute the input register (using the fact that the shifted-out bits were zero). Both of those out-of-place operations can be implemented in $O(m)$ gates, $O(m)$ space, and $O(\log m)$ depth, by using a tree of controlled-NOT gates to create m copies of the control bit and then performing two layers of m Toffoli gates between the control, input, and output registers, separated by a layer of NOT gates on the controls: the first layer XORs the output register by the shifted value of the input if the control is on, and the second layer XORs the output by the unshifted value if the control is off. Finally, the m copies of the control bit are uncomputed by another tree of $O(m)$ controlled-NOT gates.

Step 2 of Algorithm 4.1 can be implemented in $O(1)$ gates, depth, and space, because it only depends on a constant number of the bits of t and x'. Step 3 of Algorithm 4.1 requires calling Algorithm 4.2, which by Lemma 4.6 can be performed with the complexities specified in the Theorem. Step 4 consists of the computation of the Jacobi symbol of two m-bit inputs, which can be performed in $G_{\text{Jac}}(m)$, space $S_{\text{Jac}}(m)$, and depth $D_{\text{Jac}}(m)$ by supposition. Finally, step 5 can be performed with the stated complexities given that steps 3 and 1 can.

Thus all steps can be implemented reversibly with the stated complexities, completing the proof. □

Lemma 4.6. *Suppose there exists a quantum multiplication circuit on t-bit inputs with gates $G_{\text{mult}}(t)$, space $S_{\text{mult}}(t)$, and depth $D_{\text{mult}}(t)$. Then, there exists a quantum circuit implementing Algorithm 4.2 with gates $O\left(\frac{n}{m} \cdot G_{\text{mult}}(m)\right)$, space $O(S_{\text{mult}}(m))$ qubits, and depth $O\left(\left(\frac{n}{m} + \log m\right) \cdot D_{\text{mult}}(m)\right)$.*

Proof. We proceed by showing that each step of the algorithm can be performed reversibly with the stated complexity. We note that $S_{\text{mult}}(t) \geq O(m)$ because its inputs are quantum, so the overall algorithm's space is lower bounded by $O(m)$; and $D_{\text{mult}}(t) \geq O(\log m)$ because each bit of a multiplier's input can affect $O(m)$ bits of its output, so the overall depth is lower bounded by $O(\log m)$. Both bounds hold for any choice of (reversible) multiplier.

Both parts of Step 2 (and its uncomputation, Step 4) are arithmetic divisions. When implemented via Newton iteration, the gate and space complexity of division is the same as multiplication up to a constant factor [Knu98]; the depth complexity for t-bit inputs is bounded by $O(\log t \cdot D_{\text{mult}}(t))$ (although the bound improves to $O(D_{\text{mult}}(t))$ if $D_{\text{mult}}(t) \geq \Omega(t^\epsilon)$ for any $\epsilon > 0$). For step 3, each iteration of the loop consists of a constant number of additions, subtractions, and multiplications, all of size $O(m)$. The additions and subtractions can be implemented in gate count and space $O(m)$, and depth $O(\log m)$, via quantum carry-lookahead addition [DKRS06]. The multiplications can be performed with gates $G_{\text{mult}}(m)$, space $S_{\text{mult}}(m)$, and depth $D_{\text{mult}}(m)$ by supposition. The loop has $n/m - 1$ iterations, so overall, step 3 can be implemented in $O\left(\frac{n}{m} \cdot G_{\text{mult}}(m)\right)$ gates, $O(S_{\text{mult}}(m))$ qubits of space, and $O\left(\frac{n}{m} \cdot D_{\text{mult}}(m)\right)$ depth. Thus all of Algorithm 4.2 can be implemented in the stated depth, space, and gate count, completing the proof. □

4.2 Implications: Factoring Certain Integers in Sublinear Space and Depth

In this section, we instantiate Algorithms 4.1 and 4.2. Here we focus on asymptotic costs, leaving the optimization of circuits for practical problem sizes to future work. For multiplication, we use a parallelized quantum circuit for Schönhage-Strassen multiplication [SS71], by which the product of two t-bit quantum integers can be computed in gate count $\widetilde{O}(t)$ and depth $\mathsf{polylog}(t)$, using $\widetilde{O}(t)$ total qubits [NZLS23]. For computing the Jacobi symbol of two inputs of size t, there exist classical algorithms with complexity $\widetilde{O}(t)$ [Sch71, TY90, BS96, Möl08]; by standard reversible circuit techniques these algorithms can be made into quantum circuits with gate count, depth, and qubit count all at most $\widetilde{O}(t)$ [Ben73, Ben89, LS90]. The following corollary results directly from instantiating Algorithms 4.1 and 4.2 with these constructions.

Corollary 4.7 (Compare with Corollary 3.2)**.** *There exists a quantum circuit for the unitary of Equation* (5) *with gate count $\widetilde{O}(n)$, depth $\widetilde{O}(n/m + m)$, and space $\widetilde{O}(m)$ qubits.*

Consequently, by Theorem 3.1, we can recover prime P and Q (with $Q < 2^m$) from an n-bit input $N = P^2Q$ with $\widetilde{O}(n)$ gates in $\widetilde{O}(n/m + m)$ depth, using $\widetilde{O}(m)$ qubits.

Remark 5 (Near-optimal parallelism for small Q)**.** *It is possible to achieve depth $\widetilde{O}(n/m + \log Q)$ using $\widetilde{O}(m)$ qubits for any $m \geq \log Q$, by using block size m in Algorithm 4.1 but implementing step 4 via a recursive call to Algorithm 4.1 with a smaller block size (and possibly further levels of recursion if needed). This optimization becomes relevant when $\log Q < O(\sqrt{n})$, such that the n/m term in the depth could dominate. In that regime, this trick allows the depth to be reduced as low as $\widetilde{O}(\log Q)$ at the expense of qubit count $\widetilde{O}(n/\log Q)$ (by setting $m = n/\log Q$). In general, for a target depth d where $\log Q \leq d \leq n/\log Q$, at the i^{th} level of recursion the block size m_i should be set to $m_i = m_{i-1}/d$ (with $m_1 = n/d$) and the recursion stops when $m_i < d$. This construction yields a depth $\widetilde{O}(d)$ and qubit count $\widetilde{O}(n/d)$. The space-time product (qubit count times depth) is $\widetilde{O}(n)$, the same as the gate count, thus nearly achieving the asymptotically optimal limit for parallelism of $O(1)$ operations per qubit per time step (up to polylogarithmic factors).*

Our Factoring-Based Proof of Quantumness. We now state the implications of Corollary 4.7 when factoring numbers of the form $N = P^2Q$ with $\log Q = \widetilde{\Theta}((\log N)^{2/3})$ and P, Q are prime. As summarized in Section 2.3, there are no known classical special-purpose factoring algorithms that perform better than the general number field sieve [Pol93, LLMP90, BLP93] on integers of this form. Yet by Corollary 4.7, it is possible to quantumly factor these integers in much less space and depth than would be required for generic integers [Sho97, Reg25] or even generic squarefull integers [LPDS12]. Indeed, applying Corollary 4.7 to numbers of that form yields the following result:

Corollary 4.8. *Consider n-bit numbers of the form $N = P^2Q$, where P, Q are primes and $\log Q = \widetilde{\Theta}(n^{2/3})$. There exist quantum circuits for recovering P and Q from N with $\widetilde{O}(n)$ gates, $\widetilde{O}(n^{2/3})$ depth, and $\widetilde{O}(n^{2/3})$ qubits.*

5 Completely Factoring Integers with Distinct Exponents in their Prime Factorization

Here, we provide a black-box reduction that shows that any algorithm achieving the guarantees of Theorem 3.1 can in fact be used to *completely factor* integers of the form $N = p_1^{\alpha_1} \dots p_r^{\alpha_r}$ with $\alpha_1, \dots, \alpha_r$ positive and distinct.

Definition 5.1 ([AM17]). *We say that an integer N is* special *if all the exponents in its prime factorization are distinct i.e. we can write $N = p_1^{\alpha_1} \dots p_r^{\alpha_r}$ for distinct primes $p_1, \dots, p_r$ and distinct positive integers $\alpha_1, \dots, \alpha_r$.*

Remark 6. *One might wonder whether special integers turn out to be classically easy to factor as well. We state some evidence that this is not the case here. Indeed, the density of these integers was studied by Aktaş and Murty [AM17], who showed that for any integer $N_{\max}$, the number of special integers $N \in [1, N_{\max}]$ is $\approx \frac{1.7 N_{\max}}{\ln N_{\max}}$. The following classes of integers that are classically even slightly easier than general to completely factor do not contain enough elements in total to cover all special numbers:*

- *Prime numbers: there are $\approx \frac{N_{\max}}{\ln N_{\max}}$ of these by the prime number theorem.*
- *Integers of the form a^b for $b > 1$: there are at most $\widetilde{O}\left(\sqrt{N_{\max}}\right)$ of these.*
- *Sub-exponentially smooth integers (i.e. integers whose largest prime divisor is at most say $\exp\left(O((\log N_{\max})^{0.99})\right)$): there are $N_{\max} \cdot \exp\left(-\widetilde{O}((\log N_{\max})^{0.01}\right)$ of these [Gra00].*

With this in mind, we now turn our attention to completely factoring special integers. The results and ideas in this section bear some high-level similarity to previous work by Yun [Yun76] in the context of factoring polynomials. Concretely, Yun shows that any polynomial $f(x)$ can be decomposed into a factorization $f(x) = g_1(x)^{\alpha_1} \dots g_r(x)^{\alpha_r}$ where the g_i's are squarefree and pairwise coprime and the α_i's are distinct. The starting point of Yun's algorithm is the observation that if f is divisible by the square of some polynomial g, then g will divide $\gcd(f, f')$. Similarly, in this section we will start from an algorithm for calculating squarefree decompositions of integers and obtain an algorithm for fully factoring special integers.

We first begin with a simple lemma. At a high level, we want to show that the $\Omega(1)$ success probability in Theorem 3.1 can be boosted to be very close to 1. This is not obvious since given $N = A^2 B$ with B squarefree, it may not be possible to efficiently determine whether the algorithm has succeeded in recovering B. We show that this is not difficult to work around.

Lemma 5.2. *Let N be a positive integer, with unique representation as $N = A^2 B$ for B squarefree. Moreover, we say that N is* very good *if it is composite and neither squarefree nor a square.*

Assume there exists an algorithm $\mathcal{A}$ that given a very good integer N, outputs either B or a prime dividing N with probability $\Omega(1)$.

Then for any positive integer T, there exists another algorithm $\mathcal{A}'$ that given a positive integer $N = A^2 B$ with B squarefree, either outputs B or a prime divisor of N with probability $1 - \exp(-\Omega(T))$. This algorithm makes at most T calls to $\mathcal{A}$ with the same input N. Outside of calls made to $\mathcal{A}$, the algorithm is classical and runs in time $\mathsf{poly}(\log N)$.

Proof. First, we state the main idea. Suppose that $\mathcal{A}$ produces some composite B' as output. The main observation is that while we cannot efficiently check whether $B' = B$, we can efficiently check that N/B' is a square. Moreover, if B' satisfies this condition then B' must be divisible by B. With this in mind, $\mathcal{A}'$ will proceed as follows. We will use B^* to denote the algorithm's final output:

1. If N is prime, we can output N itself and terminate. If N is a square, we can easily check this and output $B^* = 1$ and terminate. Henceforth we can assume that N is either very good or squarefree.

2. Now run $\mathcal{A}$ T times and let the outputs be $B_1, \dots, B_T$. If there exists some j such that B_j is a prime divisor of N, output B_j and terminate.

3. Otherwise, let $S \subseteq \{B_1, \dots, B_T\}$ be the set of all values B' in this list such that B' divides N and N/B' is a square.

4. If $S = \emptyset$, output $B^* = N$ (this is equivalent to declaring that N is squarefree). Otherwise, output B^* as the smallest element in S and terminate.

First, suppose N is very good. In this case, at least one of the runs of $\mathcal{A}$ will be successful (i.e. it outputs B or a prime divisor of N) with probability $1 - \exp(-\Omega(T))$. Then, assuming at least one of the runs of $\mathcal{A}$ is successful, we have two cases:

- If the successful run produced a prime divisor, $\mathcal{A}'$ will detect this and output accordingly.
- If the successful run produced B, then this will be included in S. Moreover, all elements of S must be divisible by B (and hence $\geq B$). Hence taking the minimal element in S will output B.

Finally, suppose N is squarefree. In this case, we have no guarantee on the behavior of $\mathcal{A}$. But if it produces a prime divisor of N, $\mathcal{A}'$ will detect and output this. Otherwise, note that the only integer that could be included in S is N itself. So either S will be empty or its smallest element will be N, and in either case $\mathcal{A}'$ will output N. The conclusion follows. □

The below theorem and its proof bear some high-level similarity to a result by Yun [Yun76] that shows that a similar factorization can easily be carried out for polynomials.

Theorem 5.3. *Assume there exists an algorithm $\mathcal{A}$ that given a positive integer $N = A^2B$ with B squarefree and parameter T, either outputs B or a prime divisor of N with probability with $1 - \exp(-\Omega(T))$.*

Then there exists another algorithm $\mathcal{B}$ that, given a special integer N as input, recovers the complete prime factorization of N with probability $1 - \mathsf{negl}(\log N)$. This algorithm makes at most $O(\sqrt{\log N})$ calls to $\mathcal{A}$ with inputs N' that are always $\leq N$ and with repetition parameter $T = \omega(\log N)$. Outside of calls made to $\mathcal{A}$, the algorithm is classical and runs in time $\mathsf{poly}(\log N)$.

Proof. Let us write $N = p_1^{\alpha_1} \dots p_r^{\alpha_r}$. Then note firstly that since the α_i's are distinct, we have $N \geq 2^{\alpha_1 + \dots + \alpha_r} \geq 2^{\Omega(r^2)} \Rightarrow r \leq O(\sqrt{\log N})$. It hence suffices to show that we can accomplish the desired task with $O(r)$ calls.

We present our algorithm in Algorithm 5.1. The efficiency is clear since after every call to $\mathcal{A}$, the number of distinct prime divisors of M decreases by 1. As for correctness, note firstly that our procedure clearly preserves the fact that M is special at each step. Updating $M \leftarrow \sqrt{M}$ will halve all the exponents in its prime factorization which keeps them distinct. Otherwise, we take a prime and remove as many factors of it from M as possible. This effectively just removes an element from the set of nonzero exponents in the prime factorization of M, which clearly preserves distinctness.

It then remains to justify that if M is special, then with all but negligible probability $B/\gcd(k, B)$ will be prime. Write $M = \prod_{i=1}^{s} q_i^{\beta_i}$ for distinct primes q_i and distinct positive integers β_i. Then if the output B of $\mathcal{A}$ is not prime, we will have (with probability $1 - \mathsf{negl}(\log N)$) that

$$B = \prod_{i \in [s] : \beta_i \text{ odd}} q_i.$$

Now among the indices $i \in [s]$ such that β_i is odd, let i^* be the index such that β_i is minimal. Then $\gamma = \beta_{i^*}$ (where γ is defined as computed in Algorithm 5.1), and hence

$$k = \left(\prod_{i \in [s] : \beta_i \text{ even}} q_i^{\beta_i} \right) \cdot \left(\prod_{i \in [s] : \beta_i \text{ odd}} q_i^{\beta_i - \beta_{i^*}} \right).$$

The crucial point is that for any $i \in [s]$ with $i \neq i^*$ we will have $\beta_i \neq \beta_{i^*}$, because M is special. In particular, for $i \in [s]$ such that β_i is odd and $i \neq i^*$, k must be divisible by q_i. On the other hand, k is clearly not divisible by q_{i^*}. It follows that

$$\gcd(k, B) = \prod_{i \in [s] : \beta_i \text{ odd and } i \neq i^*} q_i \Rightarrow \frac{B}{\gcd(k, B)} = q_{i^*},$$

which is indeed prime. This completes our proof of the theorem.

Algorithm 5.1: Completely factoring special integers (see Theorem 5.3)

Data: A special positive integer N.

Result: A full factorization of N (with probability $1 - \mathsf{negl}(\log N)$).

1. Initialize M to be N, and initialize F to be the "empty factorization" (i.e. the factorization of 1). We will maintain the invariants that M is a special divisor of N and F is the factorization of N/M.

2. Repeat the following until $M = 1$:

   (a) If M is a prime or prime power, add the prime factorization of M to F, and update $M \leftarrow 1$ (it is well-known that this can be efficiently done classically; we sketch this in Section 2.3).

   (b) Else if M is square, recurse, calling Algorithm 5.1 on input $\sqrt{M}$; add two entries to F for each prime factor in the result. Then set $M \leftarrow 1$.

   (c) Otherwise, if neither the conditions in (a) nor (b) hold, apply algorithm $\mathcal{A}$ to M, with $T = \omega(\log N)$ so that the success probability is $1 - \mathsf{negl}(\log N)$. Now proceed as follows:

   - If the output B is prime: repeatedly divide M by B until M is not divisible by B. Update F accordingly and continue to the next step of the loop.
   - Otherwise, we can assume that B is squarefree and M/B is square (with probability $1 - \mathsf{negl}(\log N)$). Then by repeatedly dividing M by B, we can find integers k, γ such that $M = k \cdot B^\gamma$ and k is not divisible by B.
     Then compute $p = B/\gcd(k, B)$ (which can be done efficiently; see Section 2.3 for an overview of some algorithms for computing GCDs) and check whether p is prime. If it is not, abort (we will show that this almost never occurs). Otherwise, divide M by as many factors of p as possible and update F accordingly.

3. Output F.

□

Combining Corollary 3.2, Lemma 5.2, and Theorem 5.3 yields the following result:

Corollary 5.4. *A special integer N can be completely factored with success probability $1 - \mathsf{negl}(\log N)$ using $\omega((\log N)^{3/2})$ calls to a quantum circuit of size $\widetilde{O}(n)$.*

Proof. The circuit in Corollary 3.2 only requires $\widetilde{O}(n)$ gates. Then the algorithm in Lemma 5.2 can be realized with $T = \omega(\log N)$ calls to the circuit in Corollary 3.2 (here, the choice of T is specified by Theorem 5.3.) Finally, the algorithm in Theorem 5.3 can be realized with $O(\sqrt{\log N})$ calls to the algorithm of Lemma 5.2. Putting these together, the conclusion follows. □

Acknowledgements. The authors would like to thank Henry Corrigan-Gibbs for giving a stimulating talk at CRYPTO 2024 that inspired the beginning of this project, and for useful subsequent discussions. The authors would also like to thank Isaac Chuang, Antoine Joux, Mikhail Lukin, and Peter Shor for insightful discussions. The authors would also like to thank Daniel J. Bernstein, Martin Ekerå, Laura Lewis, Oded Regev, and anonymous reviewers for helpful comments and feedback on the manuscript.

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. 2

[AGGM24] Petia Arabadjieva, Alexandru Gheorghiu, Victor Gitton, and Tony Metger. Single-Round Proofs of Quantumness from Knowledge Assumptions, May 2024. 4

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, September 2004. 12

[AM17] Kevser Aktaş and M. Ram Murty. On the number of special numbers. *Proceedings - Mathematical Sciences*, 127:423–430, 2017. 8, 31

[AMMW24] Yusuf Alnawakhtha, Atul Mantri, Carl A. Miller, and Daochen Wang. Lattice-Based Quantum Advantage from Rotated Measurements. *Quantum*, 8:1399, July 2024. 3

[AZ24] Scott Aaronson and Yuxuan Zhang. On verifiable quantum advantage with peaked circuit sampling, April 2024. 3

[BBM17] Daniel J. Bernstein, Jean-François Biasse, and Michele Mosca. A Low-Resource Quantum Factoring Algorithm. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography*, pages 330–346, Cham, 2017. Springer International Publishing. 12

[BCDP96] David Beckman, Amalavoyal N Chari, Srikrishna Devabhaktuni, and John Preskill. Efficient networks for quantum factoring. *Physical Review A*, 54(2):1034, 1996. 1

[BCM+21] Zvika Brakerski, Paul Christiano, Urmila Mahadev, Umesh Vazirani, and Thomas Vidick. A Cryptographic Test of Quantumness and Certifiable Randomness from a Single Quantum Device. *Journal of the ACM (JACM)*, August 2021. 3

[BDH99] Dan Boneh, Glenn Durfee, and Nick Howgrave-Graham. Factoring $N = p^r q$ for large r. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 1999. 1, 3, 4, 12

[Bea03] Stéphane Beauregard. Circuit for Shor's algorithm using 2n+3 qubits. *Quantum Inf. Comput.*, 3(2):175–185, 2003. 1

[Ben73] C. H. Bennett. Logical Reversibility of Computation. *IBM Journal of Research and Development*, 17(6):525–532, November 1973. Conference Name: IBM Journal of Research and Development. 11, 24, 30

[Ben89] Charles H. Bennett. Time/Space Trade-Offs for Reversible Computation. *SIAM Journal on Computing*, 18(4):766–776, August 1989. Publisher: Society for Industrial and Applied Mathematics. 11, 24, 30

[BHLV17] Daniel J. Bernstein, Nadia Heninger, Paul Lou, and Luke Valenta. Post-quantum RSA. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography*, pages 311–329, Cham, 2017. Springer International Publishing. 12

[BKVV20] Zvika Brakerski, Venkata Koppula, Umesh Vazirani, and Thomas Vidick. Simpler Proofs of Quantumness. In Steven T. Flammia, editor, *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, volume 158 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 3, 4

[BL94] Johannes A. Buchmann and Hendrik W. Lenstra Jr. Approximating rings of integers in number fields. *Journal Theorie de Nombres Bordeaux*, 6:221–260, 1994. 4

[BL96] Dan Boneh and Richard J. Lipton. Algorithms for black-box fields and their application to cryptography (extended abstract). In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 1996. 5, 6

[BLP93] J. P. Buhler, H. W. Lenstra, and Carl Pomerance. Factoring integers with the number field sieve. In Arjen K. Lenstra and Hendrik W. Lenstra, editors, *The development of the number field sieve*, pages 50–94, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. 1, 11, 30

[BS96] E. Bach and J.O. Shallit. *Algorithmic Number Theory: Efficient algorithms*. Number v. 1 in Algorithmic Number Theory. MIT Press, 1996. 2, 6, 7, 9, 11, 12, 16, 22, 23, 24, 30

[BS11] Gaetan Bisson and Andrew V. Sutherland. Computing the endomorphism ring of an ordinary elliptic curve over a finite field. *Journal of Number Theory*, 131(5):815–831, 2011. Elliptic Curve Cryptography. 4

[CEMM98] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proc. Roy. Soc. Lond. A*, 454:339, 1998. 26

[CFRZ16] Jean-Sébastien Coron, Jean-Charles Faugère, Guénaël Renault, and Rina Zeitoun. Factoring n=pˆrqˆs for large r and s. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, volume 9610 of *Lecture Notes in Computer Science*, pages 448–464. Springer, 2016. 1, 4, 12

[CFS24] Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher. Reducing the Number of Qubits in Quantum Factoring, 2024. 1, 3, 12

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4):557–594, July 2004. 4

[Che24] Yilei Chen. Quantum algorithms for lattice problems. *IACR Cryptol. ePrint Arch.*, page 555, 2024. 18, 19

[CJLN09] Guilhem Castagnos, Antoine Joux, Fabien Laguillaumie, and Phong Q. Nguyen. Factoring pq^2 with quadratic forms: Nice cryptanalyses. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 469–486. Springer, 2009. 1, 4

[CL09] Guilhem Castagnos and Fabien Laguillaumie. On the security of cryptosystems with quadratic decryption: The nicest cryptanalysis. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 260–277. Springer, 2009. 1, 4

[Con] Keith Conrad. Gauss and Jacobi sums on finite fields and $\mathbb{Z}/m\mathbb{Z}$. http://kconrad.math.uconn.edu/blurbs/gradnumthy/Gauss-Jacobi-sums.pdf. 14

[Cop02] Don Coppersmith. An approximate Fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067*, 2002. 1, 16

[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. 1, 3, 5

[CW24] Henry Corrigan-Gibbs and David J. Wu. The one-wayness of jacobi signatures. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part V*, volume 14924 of *Lecture Notes in Computer Science*, pages 3–13. Springer, 2024. 5, 6

[DKRS06] Thomas G. Draper, Samuel A. Kutin, Eric M. Rains, and Krysta M. Svore. A logarithmic-depth quantum carry-lookahead adder. *Quantum Information & Computation*, 6(4):351–369, July 2006. 29

[EG24a] Martin Ekerå and Joel Gärtner. Extending Regev's factoring algorithm to compute discrete logarithms. In Markku-Juhani Saarinen and Daniel Smith-Tone, editors, *Post-Quantum Cryptography*, pages 211–242, Cham, 2024. Springer Nature Switzerland. 1

[EG24b] Martin Ekerå and Joel Gärtner. A high-level comparison of state-of-the-art quantum algorithms for breaking asymmetric cryptography. *CoRR*, abs/2405.14381, 2024. 1

[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. 1, 3, 12, 22

[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. 1, 2, 4

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

[Gid19] Craig Gidney. Asymptotically efficient quantum Karatsuba multiplication. *arXiv preprint arXiv:1904.07356*, 2019. 1

[GR02] Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions, 2002. 22

[Gra00] Andrew Granville. Smooth numbers: Computational number theory and beyond. *Math. Sci. Res. Inst. Publ.*, 44, 01 2000. 31

[HH00] Lisa Hales and Sean Hallgren. An improved quantum Fourier transform algorithm and applications. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 515–525. IEEE Computer Society, 2000. 2, 6

[HH22] David Harvey and Markus Hittmeir. A deterministic algorithm for finding r-power divisors. *Research in Number Theory*, 8(4), October 2022. 1, 4

[HJN+20] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. Improved Quantum Circuits for Elliptic Curve Discrete Logarithms. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 425–444, Cham, 2020. Springer International Publishing. 4

[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. 1

[HvdH21] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2), March 2021. 11, 23

[HW75] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford, fourth edition, 1975. 17

[KCVY21] Gregory D. Kahanamoku-Meyer, Soonwon Choi, Umesh V. Vazirani, and Norman Y. Yao. Classically-verifiable quantum advantage from a computational Bell test. *CoRR*, abs/2104.00687, 2021. 3, 4, 5

[KLVY23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Lisa Yang. Quantum Advantage from Any Non-local Game. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1617–1628, New York, NY, USA, June 2023. Association for Computing Machinery. 3

[KM15] Neal Koblitz and Alfred J. Menezes. The random oracle model: A twenty-year retrospective. *Designs, Codes and Cryptography*, 77(2):587–610, December 2015. 4

[KMY24] Gregory D. Kahanamoku-Meyer and Norman Y. Yao. Fast quantum integer multiplication with zero ancillas, 2024. 1, 11, 23

[Knu98] Donald Ervin Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998. 11, 29

[Len87] H. W. Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics*, 126(3):649–673, 1987. 1, 3, 11, 12

[LLMP90] Arjen K. Lenstra, Hendrik W. Lenstra Jr., Mark S. Manasse, and John M. Pollard. The number field sieve. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 564–572. ACM, 1990. 1, 11, 30

[LPDS12] Jun Li, Xinhua Peng, Jiangfeng Du, and Dieter Suter. An efficient exact quantum algorithm for the integer square-free decomposition problem. *Scientific Reports*, 2, 2012. 2, 3, 4, 5, 6, 12, 14, 15, 16, 22, 30

[LS90] Robert Y. Levine and Alan T. Sherman. A Note on Bennett's Time-Space Tradeoff for Reversible Computation. *SIAM Journal on Computing*, 19(4):673–677, August 1990. Publisher: Society for Industrial and Applied Mathematics. 11, 24, 30

[May10] Alexander May. Using LLL-reduction for solving RSA and factorization problems. In Phong Q. Nguyen and Brigitte Vallée, editors, *The LLL Algorithm - Survey and Applications*, Information Security and Cryptography, pages 315–348. Springer, 2010. 1, 4

[MBV20] Michele Mosca, Joao Marcos Vensi Basso, and Sebastian R. Verschoor. On speeding up factoring with quantum SAT solvers. *Scientific Reports*, 10(1):15022, September 2020. 12

[Mil24] Carl A. Miller. Hidden-State Proofs of Quantumness, October 2024. 3

[Möl08] Niels Möller. On Schönhage's algorithm and subquadratic integer gcd computation. *Math. Comput.*, 77:589–607, 2008. 2, 6, 7, 11, 12, 16, 22, 24, 30

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985. 8, 23

[Mul24] Erik Mulder. Fast square-free decomposition of integers using class groups. 2024. 1, 3, 4, 11, 12

[MY23] Tomoyuki Morimae and Takashi Yamakawa. Proofs of quantumness from trapdoor permutations. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, volume 251 of *Leibniz International Proceedings in Informatics (Lipics)*, pages 87:1–87:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 3

[NZLS23] Junhong Nie, Qinlin Zhu, Meng Li, and Xiaoming Sun. Quantum Circuit Design for Integer Multiplication Based on Schönhage–Strassen Algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(12):4791–4802, December 2023. 3, 5, 8, 11, 22, 30

[Oka90] T. Okamoto. A fast signature scheme based on congruential polynomial operations. *IEEE Transactions on Information Theory*, 36(1):47–53, 1990. 1, 4

[OU98] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 308–318. Springer, 1998. 1, 4

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979. 11

[PO96] René Peralta and Eiji Okamoto. Faster factoring of integers of a special form. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 79:489–493, 1996. 1, 4

[Pol93] J. M. Pollard. Factoring with cubic integers. In Arjen K. Lenstra and Hendrik W. Lenstra, editors, *The development of the number field sieve*, pages 4–10, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. 1, 11, 30

[PT00] Sachar Paulus and Tsuyoshi Takagi. A new public-key cryptosystem over a quadratic order with quadratic decryption time. *J. Cryptol.*, 13(2):263–272, 2000. 1, 4

[PZ03] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Inf. Comput.*, 3(4):317–344, 2003. 11

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009. 22

[Reg25] Oded Regev. An efficient quantum factoring algorithm. *J. ACM*, 72(1), January 2025. 1, 3, 5, 12, 22, 30

[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. 7

[RV24] Seyoon Ragavan and Vinod Vaikuntanathan. Space-efficient and noise-robust quantum factoring. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VI*, volume 14925 of *Lecture Notes in Computer Science*, pages 107–140. Springer, 2024. 1, 3, 5, 25

[Sch71] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1(2):139–144, 1971. 2, 3, 5, 6, 7, 11, 12, 16, 22, 24, 30

[Sch18] John M. Schanck. Multi-power post-quantum RSA. *IACR Cryptol. ePrint Arch.*, page 325, 2018. 8

[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. 1, 22

[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, 3, 4, 5, 6, 12, 16, 17, 19, 22, 30

[SS71] Arnold Schönhage and Volker Strassen. Fast multiplication of large numbers. *Computing*, 7:281–292, 1971. 8, 11, 23, 30

[SS06] Katja Schmidt-Samoa. A new Rabin-type trapdoor permutation equivalent to factoring. *Electronic Notes in Theoretical Computer Science*, 157(3):79–94, 2006. Proceedings of the First International Workshop on Security and Trust Management (STM 2005). 1, 4

[Tak98] Tsuyoshi Takagi. Fast RSA-type cryptosystem modulo $p^k q$. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 318–326. Springer, 1998. 1, 4

[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. 1

[TY90] Klaus Thull and Chee K Yap. A uni ed approach to hgcd algorithms for polynomials and integers, 1990. 2, 6, 7, 11, 12, 16, 22, 24, 30

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

[Yun76] David Y.Y. Yun. On square-free decomposition algorithms. In *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation*, SYMSAC '76, page 26–35, New York, NY, USA, 1976. Association for Computing Machinery. 2, 8, 31, 32

[YZ22] Takashi Yamakawa and Mark Zhandry. Verifiable Quantum Advantage without Structure. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 69–74, October 2022. 3

[Zal06] Christof Zalka. Shor's algorithm with fewer (pure) qubits, 2006. 1