Reducing the Number of Qubits in Quantum Factoring ⋆

Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher

Univ Rennes, Inria, CNRS, IRISA
firstname.lastname@inria.fr

Abstract. This paper focuses on the optimization of the number of logical qubits in quantum algorithms for factoring and computing discrete logarithms in $\mathbb{Z}_N^*$. These algorithms contain an exponentiation circuit modulo N, which is responsible for most of their cost, both in qubits and operations.

In this paper, we show that using only $o(\log N)$ work qubits, one can obtain the least significant bits of the modular exponentiation output. We combine this result with May and Schlieper's truncation technique (ToSC 2022) and the Ekerå-Håstad variant of Shor's algorithm (PQCrypto 2017) to solve the discrete logarithm problem in $\mathbb{Z}_N^*$ using only $d + o(\log N)$ qubits, where d is the bit-size of the logarithm. Consequently we can factor n-bit RSA moduli using $n/2 + o(n)$ qubits, while current envisioned implementations require about $2n$ qubits.

Our algorithm uses a Residue Number System and succeeds with a parametrizable probability. Being completely classical, we have implemented and tested it. For RSA factorization, we can reach a gate count $\mathcal{O}(n^3)$ for a depth $\mathcal{O}(n^2 \log^3 n)$, which then has to be multiplied by $\mathcal{O}(\log n)$ (the number of measurement results required by Ekerå-Håstad). To factor an RSA-2048 instance, we estimate that 1730 logical qubits and 2^{36} Toffoli gates will suffice for a single run, and the algorithm needs on average 40 runs. To solve a discrete logarithm instance of 224 bits (112-bit classical security) in a safe-prime group of 2048 bits, we estimate that 684 logical qubits would suffice, and 20 runs with 2^{32} Toffoli gates each.

Keywords: Quantum cryptanalysis, Shor's algorithm, Integer factoring, Discrete Logarithms, Residue number system.

1 Introduction

In 1994, Shor [34] introduced a polynomial-time quantum algorithm for factoring integers and computing Discrete Logarithms (DL). This remains to date one of the most powerful applications of quantum cryptanalysis, which caused the birth of *post-quantum cryptography*.

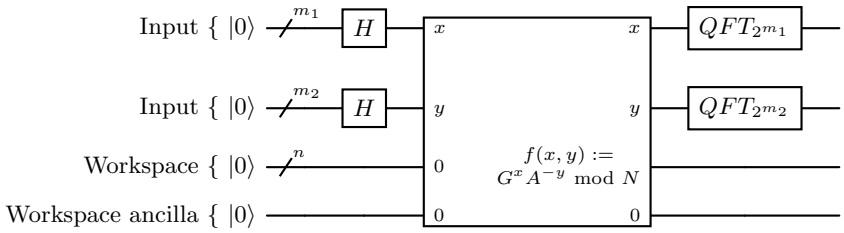

Fig. 1. Shor's quantum DL subroutine.

Shor's algorithm for DL relies on a quantum period-finding subroutine represented in Figure 1, which finds the period of a function $f : \mathcal{G} \mapsto \{0,1\}^n$ for $(\mathcal{G}, +)$ any Abelian group where operations are efficiently computable.

Let G be a multiplicative generator of $\mathbb{Z}_N^*$. In order to find the DL of A, one defines the function f over $\mathbb{Z}^2$ by $f(x,y) = G^x A^{-y} \bmod N$. Obviously the pair $(D,1)$, where D is such that $A = G^D$, is a period of f: one has $f(x+D, y+1) = f(x,y)$ for all (x,y).

The algorithm starts by initializing an *input register* x of $m := m_1 + m_2$ bits and a *workspace register*. It then produces a uniform superposition in the input x and computes f in the workspace. It then performs a Quantum Fourier Transform (QFT) on the input register again. Upon measurement, only the value of the input register is used. A classical post-processing extracts the period (hence the DL) from one or more measurement outputs. The case of factoring is simpler: one just looks for the multiplicative order of a constant A modulo N, so the function is defined over $\mathbb{Z}$ by $f(x) = A^x \bmod N$.

For both factoring and DL, one needs only $m = \mathcal{O}(\log N)$. Besides, both the Hadamard and Fourier transform can be performed in place and efficiently, meaning that the computational cost is mostly determined by the cost of the modular exponentiation circuit: computing $G^x A^{-y} \bmod N$. Like previous works, this is the part that we target. This operation can entirely be implemented using classical reversible logic, i.e., Toffoli, CNOT and NOT gates. This is what we do in this paper, although many previous works have also used "inherently quantum" arithmetic circuits based on the QFT and phase operations [9,24].

Optimization of Space. Since Shor's original paper, precisely estimating the cost of the algorithm has remained a crucial question. Many authors have designed circuits optimizing either its qubit or gate count [3,38,36,22,16].

For factoring, most of these works report a qubit count of $2n + 1$ or above, where $n = \log_2 N$ (see for example [16] for a comparison). The best count is from [38], where Zalka proposes a circuit with $1.5n$ qubits, and suggests that it could be reduced further. However, to the best of our knowledge, this circuit has not been benchmarked in practice.

All of the recent implementations use the semi-classical Fourier transform [20] to save the space used by the input register, by reducing it to a single qubit

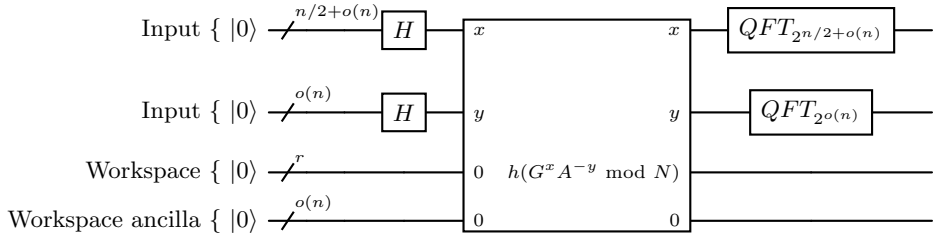

Fig. 2. New factoring subroutine: "compressed Ekerå-Håstad".

which is repeatedly measured and reused. Thus, the space cost comes entirely from modular exponentiation.

The goal of these optimizations is to reduce the physical resources required to factor large RSA moduli. For example, in [19] Gidney and Ekerå used a quantum circuit with around $3n$ logical qubits and $0.3n^3$ Toffoli gates. For RSA-2048, they estimated that 20 million physical qubits would be necessary – but only 8 hours of wall-clock time. Therefore, the number of qubits seems to be one of the strongest constraints on the physical implementation of this algorithm.

The Compression Technique. May and Schlieper studied the behavior of Shor's subroutine if the workspace register is reduced to a fraction of its size, down to a single bit [25]. This means that the circuit does not implement $(x, y) \mapsto G^x A^{-y} \bmod N$ anymore, but $(x, y) \mapsto h(G^x A^{-y} \bmod N)$ where h is a good hash function mapping $\mathbb{Z}_N^*$ to $\{0,1\}^r$, where $r \ll n$ is a small constant. They showed that if h is picked from a universal hash function family, then the "compressed" version of Shor's algorithm behaves similarly as the original one, and outputs almost the same distribution of values for the input register.

Unfortunately, this can only improve the space complexity if $h(G^x A^{-y} \bmod N)$ can be implemented with less space than naively computing $G^x A^{-y}$, and then h. This was left as an open question.

As noticed by May and Schlieper, the compression technique could be particularly useful in combination with the Ekerå-Håstad algorithm for short discrete logarithms [15]. This variant of Shor's algorithm uses multiple runs and a more involved post-processing to minimize the input register size. It finds d-bit DLs using $d + o(d)$ input qubits, and factors n-bit RSA moduli with $\frac{n}{2} + o(n)$ input qubits. Therefore, a "compressed" variant could significantly reduce the total number of qubits for both DL and factoring. Notice that this changes fundamentally the structure of the circuit, represented in Figure 2. The semi-classical Fourier transform cannot be used anymore. The input register, of size $\frac{n}{2} + o(n)$, must be kept along, and the workspace contains $o(n)$ qubits. On the positive side, this means that no intermediate measurements need to be performed.

Contributions. In this paper, we introduce a dedicated method to compute directly this "compressed" result of the modular exponentiation step in Shor's algorithm. More precisely, we compute the r least significant bits of $G^x A^{-y} \bmod N$,

instead of the complete n-bit output. Our method is completely classical. We implemented it, and designed corresponding reversible logic circuits.

Theorem (informal, see Theorem 2) *Let $m \leq 2n$ be the total size of the input register and r be a constant. For any constant $\varepsilon > 0$, there exists a reversible logical circuit for the operation $x, y \mapsto (G^x A^{-y} \bmod N) \bmod 2^r$. It succeeds with probability $1 - \varepsilon$ for inputs x, y chosen uniformly at random. It uses $\mathcal{O}\big(nm^2\big)$ gates, depth $\mathcal{O}\big(nm \log^3 n\big)$ and $\mathcal{O}\Big(\log n + \frac{m}{\log n}\Big) = o(n)$ ancilla qubits.*

Equipped with this tool, we run the Ekerå-Håstad algorithm by replacing the function $(x, y) \mapsto (G^x A^{-y} \bmod N)$ by $(x, y) \mapsto \beta \cdot ((G^x A^{-y} \bmod N) \bmod 2^r)$ for a randomly chosen mask β (the role of β is only technical here; it serves to randomize the compression function when running the theorem). We use a first heuristic which assumes that the function $(x, y) \mapsto G^x A^{-y} \bmod N$ behaves randomly (apart from its periodicity).

Heuristic 1. When N is selected at random, the function $f : x, y \mapsto G^x A^{-y} \bmod N$ behaves statistically like a random periodic function $\{0,1\}^m \to \mathbb{Z}_N^*$.

We stress that the function f is in fact not random at all, since it is trivially malleable. This heuristic is merely a guarantee that the outputs of the function are independent of the inner workings of the period-finding subroutine. It prevents a pathological situation in which the outputs of f having the same first r bits would badly interfere, breaking down the algorithm. Similar heuristics are very common in classical cryptanalysis. For example, Pollard's rho method [28] assumes the randomness of the function $x \mapsto x^2 \bmod N$ for some modulus N.

Under this heuristic, we prove that the distribution of outputs of the modified quantum subroutine is analogous to that of the uncompressed Ekerå-Håstad algorithm. More precisely, the analysis of Ekerå-Håstad [15] relies on the probability to measure some "good pairs" (x, y) in the input register, which is shown to be at least $1/8$. We show:

Theorem (informal, see Theorem 3) *Under Heuristic 1, using $r = 22$ bits for truncation, the modified Ekerå-Håstad subroutine outputs 0 with probability around $\frac{1}{2} + 0.01$ and outputs a good pair with probability at least $\frac{1}{16} - 0.01$.*

Note that this situation differs from May and Schlieper's work, which assumes that a universal hash function family $\mathbb{Z}_N^* \to \{0,1\}^r$ is given, and one can sample a new function from this family at each call to the subroutine. In our case, we want to rely only on a single function computing r bits of the output. The similarity with May and Schlieper's approach lies only in the random choice of mask β, which is important for the proof of Theorem 3.

Probability of Error. Like state-of-the art implementations of quantum exponentiation [19], our circuit is not exact. Most of the error stems from the truncation of a sum, which is very similar to the principle of oblivious carry runways [17].

Essentially, the latter allows to compute a sum faster by reasoning independently on several windows of bits, which remain independent with high probability (unless a long carry propagation occurred, which is very unlikely). In our case, we only need to compute a small window of the output bits, and we succeed for the same reason. This will be detailed in Section 4. In order to prove the correctness of our algorithm, we formulate this as a heuristic.

Heuristic 2. For a given N, G and A, our computation of $(G^x A^{-y} \bmod N) \bmod 2^r$ returns the correct result in all but a fraction $\varepsilon > 0$ of cases, where ε is a constant that can be chosen arbitrarily small.

Using a sufficiently small constant ε, the quantum subroutine will proceed without being disrupted, and the entire algorithm will succeed. For a good trade-off, the total number of runs (i.e., measurements) in the Ekerå-Håstad algorithm should be around $\mathcal{O}(\log n)$.

Theorem (Informal) *Under the two heuristics above, Algorithm 2 allows to factor a n-bit RSA modulus with a gate count in $\mathcal{O}\big(n^3 \log n\big)$ (in $\mathcal{O}(\log n)$ runs of a circuit of size $\mathcal{O}\big(n^3\big)$), depth in $\widetilde{\mathcal{O}}\big(n^2\big)$ and $\frac{n}{2} + o(n)$ qubits.*

The case of discrete logarithms in $\mathbb{Z}_P^*$, where P is a large prime, is also interesting for us. Very often, the group size is taken sufficiently large to make the number field sieve computationally infeasible, while the DL bit-size d remains quite small. The Ekerå-Håstad algorithm is particularly suited to this setting: the sub-routine has gate count $\widetilde{\mathcal{O}}\big(d^2 \log P\big)$ and uses $d + o(d) + \mathcal{O}(\log P)$ qubits.

Method. Our method is inspired by results of circuit complexity and optimizations of circuit depth using Residue Number Systems (RNS)[1]. An RNS represents a large number by a collection of residues modulo a set of small primes. In our case, the number that we represent is a *multi-product*, i.e., the product of multiple precomputed integers modulo N corresponding to powers of a constant. Using the RNS, we find that the output bits that we wish to compute can be ultimately expressed as a large sum of integers over $\mathcal{O}(\log n)$ bits. These integers are precomputed, and the bits which control the sum are computed on the fly by reducing the multi-product modulo the small primes of the RNS.

More precisely, we start from the well-known reduction of an exponentiation (with a fixed number and a variable exponent) to a controlled multi-product. Having precomputed powers of G and A, the output of the exponentiation function can be computed as: $\prod_i A_i^{z_i} \bmod N$ where the A_i are powers of the form G^{2^i} or A^{2^j} and the z_i are bits of x or y respectively. This is traditionally computed as a sequence of products by the A_i, controlled by the bits z_i; hence there are m such products, where m is the input register size defined above.

[1] A more detailed discussion on the relation between this question and circuit complexity is given in Section A.

In our algorithm, we focus on the large number $\prod_i A_i^{z_i}$, which has the order of mn bits, and represent it using the RNS. By the Chinese Remainder Theorem, this number can be expressed as a sum of residues modulo small primes, multiplied by large constants which can be precomputed[2]. Essentially the computation of residues will dominate our algorithm: there are $\mathcal{O}(nm/\log n)$ of them and they are of size $\mathcal{O}(\log n)$ bits.

To compute the modular reduction of $(\prod_i A_i^{z_i}) \bmod N$ modulo 2^r, we essentially need to compute the quotient $\lfloor \prod_i A_i^{z_i}/N \rfloor$ and reduce it modulo 2^r. We re-express this using Barrett's reduction, replacing the division by N by a multiplication followed by a binary shift. This shift complies with the reduction modulo 2^r, and our problem becomes: to compute some bits of a large integer that has been expressed as a sum of $\mathcal{O}(nm/\log n)$ integers.

Fortunately, the propagation of the carries in a sum is quite limited. In our case, we can expect carries to propagate only on about $\mathcal{O}(\log(nm))$ bits. In other words, by taking a window of $\mathcal{O}(\log(nm))$ bits, we can compute any bit of the sum with a large probability of success. More generally, we can compute r bits of the exponentiation output using $r + \mathcal{O}(\log nm)$ bits of intermediate storage.

In the case of factoring, one has $m = \mathcal{O}(n)$ and the gate complexity of our circuit becomes $\mathcal{O}\big(n^3\big)$, which is comparable to an implementation of Shor's algorithm with schoolbook multiplication. If a more efficient multiplication algorithm is used, then Shor's algorithm performs better. So does Regev's recent proposal [31]. In the case of short discrete logarithms of bit-size d in a safe-prime group $\mathbb{Z}_P^*$, we have $m = d$ by definition, which leads to a gate complexity $\widetilde{\mathcal{O}}\big(d^2 \log P\big)$. This is actually uncomparable with variants of Shor's algorithm and improves significantly over schoolbook multiplication.

Output Compression. As mentioned above, we combine the compression result of May and Schlieper [25] with the Ekerå-Håstad algorithm. However, May and Schlieper reason on average over compressing functions, while we can rely only on a single one (the truncation).

Fortunately, assuming the randomness of the exponentiation allows a reasoning quite close to the one of May and Schlieper. We compare the final quantum states between the uncompressed period-finding subroutine, and a compressed one in which the function f is also composed with a random permutation of $\mathbb{Z}_N^*$. We then prove that, on average over this random permutation, the probability to measure a given pair (x, y) (or a fixed set of such pairs) does not deviate too much from half of the previous one. The *ad hoc* constant $r = 22$ follows essentially from the constant factors in the proof, and it could likely be optimized further.

Results and Comparisons. We implemented the main components of our algorithm in order to estimate its gate and qubit counts precisely. For RSA-2048, we estimate the Toffoli gate count at $2^{35.5}$ for a single circuit, for 1730 qubits

[2] The sum needs also to be reduced modulo the product of all RNS primes; we omit this step here for ease of presentation.

including ancillas. This can be compared to Gidney and Ekerå's estimations of $2^{31.3}$ Toffoli gates and 6144 qubits [19]. However, while their algorithm succeeds in a single run, our version of Ekerå-Håstad requires multiple runs: on average 40 for RSA-2048. This brings the total Toffoli gate count to $2^{40.9}$. For a safe-prime group $\mathbb{Z}_P^*$ where P is a prime of 2048 bits, a DL instance where the logarithm has 224 bits should have 112 bits of classical security. Here our implementation uses 684 qubits, the circuit contains $2^{32.2}$ Toffoli gates and must be run 20 times. The code of our experiments is available at:

https://gitlab.inria.fr/capsule/quantum-factoring-less-qubits

Outline. In Section 2, we provide basic preliminaries of notation, arithmetic and quantum and reversible circuits. Section 3 introduces Shor's algorithm in more detail, recalls the Ekerå-Håstad variant and the May-Schlieper compression technique. Our new algorithm for computing $(G^x A^{-y} \bmod N) \bmod 2^r$ is detailed in Section 4, purely as a classical reversible computation. Then, in Section 5, we give the complete factoring algorithm and prove our main result (Theorem 4). We give some details on our cost estimates in Section 6. In Section B, we give a proof of Theorem 3 and in Section C, we give tables of parameters and resource estimates for RSA moduli and DL instances in $\mathbb{Z}_P^*$.

2 Preliminaries

In this section, we give some useful notation and basic preliminaries of arithmetic and quantum algorithms.

2.1 Notation

Let $n \in \mathbb{N}$ be the complexity parameter, typically the bit-size of the RSA modulus N we are trying to factor. From Section 4 onwards, we will work with integers of bit-size either polylogarithmic in n (typically $\mathcal{O}(\log n)$), which will be denoted by lowercase letters, or bit-size polynomial in n (typically from $\mathcal{O}(n)$ to $\mathcal{O}(n^2)$ bits), for which we use uppercase. This emphasizes that the former take negligible space w.r.t. the latter.

We use $[A]_p$ to denote $A \bmod p$, i.e., the remainder in the Euclidean division of A by p. We use $(A)_i$ to denote the i-th bit of A, where bit 0 is the least significant bit, so that: $A = \sum_i (A)_i 2^i$. Finally, given $x \in \mathbb{R}$, $\{x\} := x - \lfloor x \rfloor$ denotes the fractional part of x.

2.2 Modular Reduction and RNS

At one point in our algorithm, we need to perform modular reduction of a large number by a large modulus. For this we use a specific form of Barrett reduction.

Proposition 1. *Let N be a fixed modulus. Let $A \in \mathbb{N}$ that is not a multiple of N, and $k \in \mathbb{N}$ such that $2^k \geq AN$. Then we have:*

$$\left\lfloor \frac{A}{N} \right\rfloor = \left\lfloor \frac{A \left\lfloor 2^k/N \right\rfloor}{2^k} \right\rfloor \ . \tag{1}$$

Proof. The proof follows trivially from the fact that $x \geq \lfloor x \rfloor > x - 1$ for all x. First, we have $\left\lfloor \frac{A\left\lfloor 2^k/N \right\rfloor}{2^k} \right\rfloor \leq \frac{A}{N}$, but also $\left\lfloor \frac{A\left\lfloor 2^k/N \right\rfloor}{2^k} \right\rfloor \leq \lfloor \frac{A}{N} \rfloor$, since it's an integer. Next:

$$\forall A < 2^k, \left\lfloor \frac{A \left\lfloor 2^k/N \right\rfloor}{2^k} \right\rfloor > \frac{A \left\lfloor 2^k/N \right\rfloor}{2^k} - 1 > \frac{A}{N} - \frac{A}{2^k} - 1 \geq \left\lfloor \frac{A}{N} \right\rfloor - 2 \ .$$

This implies that the approximated quotient can be underestimated by at most 1. However, if we make the additional assumption that A is not a multiple of N, then we have: $\frac{A}{N} \geq \lfloor \frac{A}{N} \rfloor + \frac{1}{N}$. Therefore:

$$\left\lfloor \frac{A \left\lfloor 2^k/N \right\rfloor}{2^k} \right\rfloor > \left\lfloor \frac{A}{N} \right\rfloor + \frac{1}{N} - \frac{A}{2^k} - 1 \ . \tag{2}$$

In particular, setting $2^k \geq AN$ gives us $\left\lfloor \frac{A\left\lfloor 2^k/N \right\rfloor}{2^k} \right\rfloor > \lfloor \frac{A}{N} \rfloor - 1$ which implies the equality. □

Residue Number System (RNS). RNS are a well-known method of circuit optimization, typically used to obtain small-depth circuits. We emphasize that our goal here is orthogonal: to obtain circuits with potentially large depth but small memory.

A RNS uses a basis of ℓ prime moduli, which we denote by $\mathcal{P} := \{p_1, \dots p_\ell\}$, and represents a large integer A by its residues modulo the p_i. Indeed, let $M = \prod_{p \in \mathcal{P}} p$, then by the Chinese Remainder Theorem we know that there is a bijection:

$$[A]_M \mapsto [A]_{p_1}, \dots, [A]_{p_\ell} \tag{3}$$

Furthermore, this bijection can be effectively computed as follows. For each $p \in \mathcal{P}$, let $M_p = M/p$ and $w_p = (M_p)^{-1} \bmod p$. Then for any $A < M$:

$$A = \left[\sum_{p \in \mathcal{P}} [A]_p w_p M_p \right]_M \ . \tag{4}$$

The Prime Number Theorem gives us the asymptotic behavior of the prime counting function $\pi(x)$ (the number of primes smaller than x): $\pi(x) \simeq \frac{x}{\log x}$. This means that, asymptotically, $\pi(x) = \frac{x}{\log x} + o\big(\frac{x}{\log x}\big)$ and the total number of primes between x and $2x$ is of order $\frac{2x}{\log(2x)} - \frac{x}{\log x} = \mathcal{O}\big(\frac{x}{\log(x)}\big)$.

A consequence of this fact is that for any integer n, there exists a set of $\mathcal{O}\big(\frac{n}{\log n}\big)$ of primes of size $\mathcal{O}(\log n)$ bits, which can be used to represent numbers of n bits.

Lemma 1 (Consequence of the Prime Number Theorem). *For any integer $n \geq 29$, there exists a set of prime numbers $p_1, \ldots, p_\ell$ such that $\forall i, p_i < n$ and $\prod_i p_i > 2^n$. Furthermore $\ell = \mathcal{O}(n/\log n)$.*

Explicit CRT. It seems to be folklore knowledge that the quotient in the Euclidean division by M, which appears during the recombination step, can also be computed from the residues with the following formula:

$$q_M := \left\lfloor \sum_{p \in \mathcal{P}} [A]_p \frac{w_p M_p}{M} \right\rfloor = \left\lfloor \sum_{p \in \mathcal{P}} [A]_p \frac{w_p}{p} \right\rfloor \quad .$$

While early works [26] suggest to simply use floating-point arithmetic with sufficiently low rounding error, Bernstein [5,6] gives an explicit formula to compute the quotient with fixed-point arithmetic. We will follow very similar steps in Section 4.2. Although our formula differs slightly from Lemma 3.1 in [5] (and Theorem 2.2 in [6]), it would have been possible to use this Lemma immediately at this step.

2.3 Quantum Algorithms

We describe quantum algorithms using the quantum circuit model, and refer to [27] for a detailed definition including qubits, quantum states, the ket $|\cdot\rangle$ notation and the Euclidean distance $\|\cdot\|$. We emphasize that we study only *logical* quantum circuits. We do not analyze how the obtained circuits should be mapped to a physical architecture, with additional costs of routing and distillation of magic states for certain gates.

We analyze the costs of our circuits as follows. The *width* is the number of qubits required. This always includes *ancilla* qubits, which are qubits whose state is initialized to $|0\rangle$ and returned to $|0\rangle$ afterwards. The *depth* is the amount of time steps required to run the circuit if independent gates are applied in parallel. The *gate count* is the total number of gates. While this work is motivated by quantum algorithms, the circuits that we describe use only classical reversible logic, for which NOT, CNOT and Toffoli form a universal set of gates.

Finally, our factoring algorithm requires several measurement results: the same circuit is run multiple times (we use alternatively the terms of "multiple runs" or "multiple measurements" since these are the same for us). Similarly to Regev's trade-off on quantum factoring [31], it is often considered more practical to separate a quantum computation into more sub-computations of smaller complexities, as long as the entire algorithm is sufficiently robust to errors.

3 Preliminaries on Shor's Algorithm

This section gives background on Shor's algorithm [34], the Ekerå-Håstad variant [15], the precise analysis of [10] and its compressed version [25].

3.1 Shor's Algorithm

Shor's algorithm is a quantum algorithm to solve the *Abelian hidden period problem*. Consider an Abelian group $(\mathcal{G}, +)$ where operations are efficiently computable. Let $f : \mathcal{G} \mapsto \{0,1\}^n$ be a function such that $\forall x, f(x+R) = f(x)$ for some *hidden period* R. The goal is to find R.

This generic period-finding view can be specialized for the factoring and discrete logarithms (DL) cases. However, the Ekerå-Håstad algorithm reduces factoring to an instance of DL, which is why we focus on the DL case.

Shor's Algorithm for DL. We work in the multiplicative group $\mathbb{Z}_N^*$, where N is either the RSA semiprime to factor or a prime number. Let G be a generator of $\mathbb{Z}_N^*$ and let $n = \lceil \log_2 N \rceil$. Consider an element A such that $A = G^D$, where D is the unknown discrete logarithm.

We define the function f on $\mathbb{Z}^2$ as: $f(x,y) = G^x A^{-y} \bmod N$. We can observe that $\forall x, y, f(x,y) = f(x+D, y+1)$, meaning that f has a period $(D, 1)$. In fact, it has additional periods forming a two-dimensional lattice of $\mathbb{Z}^2$. We can run Shor's algorithm in two scenarios. Either we start without prior information on this lattice (aside from the expected size of D), or we start by computing the order of G and A modulo N, allowing us to restrict f to a finite group.

We define the following subroutine Q_f^{shor}:

1. Initialize an *input register* of size $m = m_1 + m_2$ and a *workspace register* of size n (not including ancilla qubits): $|0^{m_1}\rangle |0^{m_2}\rangle |0^n\rangle$
2. Create a uniform superposition in the input register, using Hadamard gates: $\frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^{m_1}-1} \sum_{y=0}^{2^{m_2}-1} |x, y\rangle |0^n\rangle$
3. Compute f into the output register: $\frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^{m_1}-1} \sum_{y=0}^{2^{m_2}-1} |x, y\rangle |f(x,y)\rangle$
4. Apply a Fourier transform $QFT_{2^{m_1}} \otimes QFT_{2^{m_2}}$ on the input register
5. Measure and return the value of the input register

Shor's Algorithm for Order-finding and Factoring. To find the multiplicative order R of A in $\mathbb{Z}_N^*$, one simply defines the function $f(x) = A^x \bmod N$. The final measurement outputs a value x which is close to some multiple of $\frac{2^m}{R}$. By choosing m large enough, R can be found from a constant number of measurements using a classical post-processing based on continued fractions. Shor proposed $m = 2n$ [34], where n is the bit-size of N.

If the order R is even, one has: $A^R = 1 \bmod N \implies (A^{R/2} - 1)(A^{R/2} + 1) = 1 \bmod N$. Assuming that $A^{R/2} - 1 \neq 0 \bmod N$, it suffices to compute a GCD between $A^{R/2} - 1$ and N to find one of its prime factors (since we focus on RSA keys, this is enough for us). Shor proved that the probability of achieving such a split is at least $1 - 2^{1-k}$, when N admits k distinct odd prime factors, which is $\geq \frac{1}{2}$ for RSA integers.

Shor's Algorithm for DL. In the DL case, one may first find the order of the elements in $\mathbb{Z}_N^*$, and restrict the function to a finite group: we will have roughly $m_1 = m_2 = n$ and the QFT operations become the QFT in this group. The post-processing operation becomes also much simpler.

Multi-Product Subroutine. Whichever the size of the input registers, the main computational cost in Shor's DL algorithm comes from implementing the exponentiation $f(x,y) := G^x A^{-y} \bmod N$. Because G and A are constants, this can be simplified into a *multi-product* subroutine.

The pair of inputs: $x, y := \sum_{i=0}^{m_1-1} x_i 2^i, \sum_{i=0}^{m_2-1} y_i 2^i$ can be identified with a pair of (m_1, m_2)-bit strings, or a single m-bit string $(x_0, \ldots, x_{m_1-1}, y_0, \ldots, y_{m_2-1})$. The output can be rewritten as:

$$f(x,y) = G^{\sum_{i=0}^{m_1-1} x_i 2^i} A^{-\sum_{i=0}^{m_2-1} y_i 2^i} \bmod N = \prod_{i=0}^{m_1-1} (G^{2^i})^{x_i} \prod_{i=0}^{m_2-1} (A^{-2^i})^{y_i} \bmod N$$
$$= \left(\prod_{i=0}^{m_1-1} (G^{2^i} \bmod N)^{x_i} \prod_{i=0}^{m_2-1} (A^{-2^i} \bmod N)^{y_i} \right) \bmod N \ ,$$

where both $(A^{-2^i} \bmod N)$ and $(G^{2^i} \bmod N)$ can be precomputed in polynomial time. The implementation of f is therefore reduced to a *multi-product* circuit:

$$z_0, \ldots, z_{m-1} \mapsto \prod_{i=0}^{m-1} A_i^{z_i} \bmod N \ , \tag{5}$$

where the A_i are either powers of A or powers of G, and the z_i are either bits of x or y. This circuit is usually implemented by a sequence of m controlled modular multiplications by the A_i. To date, no efficient circuit is known for performing such a multiplication using less than $2n$ qubits.

Semiclassical Fourier Transform. The *semiclassical Fourier transform* is a way to insert the measurement of each bit of the input register between the operations of the Fourier transform $QFT_{2^{m_1}} \times QFT_{2^{m_2}}$. In the multi-product, each bit of the input is used only once, as a control for a modular multiplication. In that case it is possible to measure it *before* initializing and using the next bit. This is why the input register can be replaced by a single qubit *if* we compute the multi-product sequentially.

3.2 Ekerå-Håstad Variant: Optimizing the Input Register Size

The Ekerå-Håstad algorithm [15] is a variant of Shor's which reduces the number of input qubits for computing short discrete logarithms. This optimization can be used in the integer factorization problem for RSA moduli, as we recall below.

Let $d \leq n$ be an upper bound on the bit-size of the DL. The Ekerå-Håstad subroutine has the same definition as Q_f^{shor} defined above, except that the number of input qubits is set appropriately. One uses $d + \ell$ qubits in the register for x and ℓ qubits in the register for y, where $\ell := \lceil \frac{d}{s} \rceil$ is much smaller than d, and the ratio s is chosen appropriately.

Different from Shor's algorithm, one runs the subroutine a total of $\mu > s$ times, followed by a lattice-based classical post-processing of the measurement

outputs. Such a procedure is presented and analyzed in detail in [10]. For the case where one wants to do a single run, a refined analysis is given in [13], but we are interested in increasing s as much as possible to reduce the space, and therefore, we consider multiple runs.

In [10], Ekerå shows that the number of measurements necessary tends asymptotically to $s+1$ when s is fixed and n tends to infinity. More precisely, for the post-processing it should be enough to perform $s+1$ runs and solve an instance of the Closest Vector Problem in a lattice of dimension $s+c$, where c is some constant, and $c=1$ is expected to be enough asymptotically. By using $s=\mathcal{O}(\log d)$ we ensure that this post-processing can be performed in polynomial time in d, using a lattice sieving algorithm of complexity $2^{\mathcal{O}(\log d)} = \mathsf{poly}(d)$.

Ekerå-Håstad for RSA Factorization. This method yields an interesting optimization in the factorization of RSA semiprimes. Consider $N = P \times Q$ where N is of bit-size n, but P and Q are both of bit-size $n/2$. Let $\tilde{P} = \frac{P-1}{2}$ and $\tilde{Q} = \frac{Q-1}{2}$. Select a random element G invertible modulo N and let R be its order: $G^R = 1 \mod N$. Then R divides $2\tilde{P}\tilde{Q}/\gcd(\tilde{P},\tilde{Q})$.

We define $f(N) = (N-1)/2 - 2^{n/2-1} = 2\tilde{P}\tilde{Q} + \tilde{P} + \tilde{Q} - 2^{n/2-1}$. Let $A := G^{f(N)} = G^D$ for some $D < R$. Then: $D = f(N) \mod R = \tilde{P} + \tilde{Q} - 2^{n/2-1} \mod R$. This means that the discrete logarithm D is small: $n/2$ bits instead of n.

Therefore, by computing the discrete logarithm of A, one obtains $\tilde{P} + \tilde{Q} - 2^{n/2-1} \mod R$, which we can assume to be equal to $\tilde{P} + \tilde{Q} - 2^{n/2-1}$, i.e., one obtains $P + Q$. Since $N = PQ$ is also known, we can deduce P and Q.

This means that for factoring RSA moduli of n bits, the input register can be made as small as $\frac{n}{2} + \mathcal{O}\left(\frac{n}{\log n}\right) = \frac{n}{2} + o(n)$ if we use $s = \mathcal{O}(\log n)$. Note that up to date, this method was not used as a space optimization, since the semiclassical Fourier transform already replaced the input register by a single qubit. However, it yields a quantum-classical trade-off and allows to optimize the number of gates.

3.3 Compressing the Subroutine

May and Schlieper [25] studied the behavior of quantum period-finding algorithms, among which Shor's and Simon's [35], when the output of the function f is *post-processed* by a hash function which reduces its size. They observed that the algorithm works almost in the same way, even if h compresses the output down to a *single bit*. This is due to the following properties.

Definition 1. *A hash function family $\mathcal{H}_t = \{h : \{0,1\}^n \to \{0,1\}^t\}$ is universal if for all $x, y \in \{0,1\}^n, x \neq y$, one has:* $\Pr_{h\in\mathcal{H}_t}[h(x) = h(y)] = 2^{-t}$.

Theorem 1 (Theorem 7 in [25]). *Let $f : \{0,1\}^m \to \{0,1\}^n$ and $\mathcal{H}_t$ be a universal hash function family. Let Q_f^{period} be a quantum circuit that, before measurement, on input $|0^m\rangle|0^n\rangle$ yields a superposition:*

$$|\Phi\rangle = \sum_{y\in\{0,1\}^m} \sum_{f(x)\in Im(f)} w_{y,f(x)} |y\rangle |f(x)\rangle \tag{6}$$

satisfying

$$\forall y \neq 0, \sum_{f(x) \in Im(f)} w_{y,f(x)} = 0 \ . \tag{7}$$

Let $p(y)$, resp. $p_h(y)$, be the probability to measure $|y\rangle, y \neq 0$ in the m input qubits when applying Q_f^{period}, resp. $Q_{h \circ f}^{\text{period}}$ with h selected u.a.r. from $\mathcal{H}_t$. Then $p_h(y) = (1 - 2^{-t})p(y)$.

Equation 7 is called the *cancellation criterion*. This criterion is satisfied by the subroutine Q_f^{shor}, but more generally by quantum period-finding of the same family, like Simon's [35] and Ekerå-Håstad.

Lemma 2 (Lemma 6 & 7 in [25]). *The Q_f^{shor} subroutine satisfies the cancellation criterion (Equation 7).*

When this property is satisfied, Theorem 1 shows that we can hash the output register. Measuring the input register then yields a similar distribution as before, where the probability to measure 0 increases (up to 1/2 when $t = 1$) and the other probabilities are rescaled accordingly. Measuring 0 reveals nothing about the period; we can just discard these results and do more measurements to succeed. However, how to compute $h \circ f$ in a space-efficient way (without essentially computing f, then h) was left as an open question in [25].

New Compression Theorem. May and Schlieper [25] remarked that the requirement of a universal hash function family may be relaxed in practice. However, with a single hash function, it seems difficult to show that $p_h(y)$ is close to $\frac{1}{2}p(y)$ for a given nonzero y; indeed, the proof of Theorem 1 largely relies on the fact that we can reason on average on the function h.

To adapt this strategy for our case, we prove a new "compression" result. We select a parameter $r > 0$. We compress the output of f by truncating it to the first r bits, then taking the dot-product with a random mask β. The mask is chosen uniformly at random from all non-zero bit-strings in $\{0,1\}^r$ before each call to the compressed subroutine. Using $r = n$ and a mask β of n bits would bring us back to Theorem 1, as this would form a universal hash function family with one-bit output.

The heuristic that we need to use is Heuristic 1. Essentially, when considering an instance of the DL or factoring problem, the function f behaves as a random periodic function $\{0,1\}^m \to \mathbb{Z}_N^*$. When r is a large enough constant, we prove in Section B that a similar property as Theorem 1 is obtained. That is, the distribution of outputs is similar to the one in the uncompressed algorithm, and its analysis can be reused.

3.4 Compressing with Approximate Arithmetic

It has been observed in previous works that Shor's algorithm does not require an exact modular exponentiation circuit. In fact, it is enough to return a good result with large probability on a random instance. Zalka first proposed such a

strategy in [37] and coined the term "deterministic errors". Further, the *coset representation of modular integers* [38] is both approximate and inherently quantum, as it represents integers modulo N using a superposition. In [17], Gidney introduced the technique of *oblivious carry runways*. These techniques have been used in state-of-the-art resource estimates of quantum factoring [19].

In our case, we will have an erroneous circuit computing a "compressed" version of the output, and we need multiple measurements. From now on, we write h instead of $h \circ f$ for the "compression" of f which truncates it to r bits.

Instead of implementing h, we can only implement a function h' such that $h'(x) = h(x)$ with probability $1 - p$ only. We do not make any assumption as to how the errors are distributed among the inputs.

Lemma 3. *Let $\mathcal{A}_i$ be an algorithm that uses a compressed period-finding subroutine with the ideal function h, makes μ measurements, post-processes the outputs, and succeeds with probability p_i. Let $\mathcal{A}_r$ be the same algorithm replacing h by h'. Then $\mathcal{A}_r$ succeeds with probability at least: $p_r \geq p_i - 8\sqrt{\mu}\sqrt{p}$.*

Proof. We start by bounding the difference between the "real" and "ideal" runs of compressed-period subroutine, using the Euclidean distance between quantum states before the last QFT operation and measurement. Indeed, if we define:

$$|\psi_r\rangle = \frac{1}{2^{m/2}} \sum_{z=0}^{2^m-1} |z\rangle\,|h'(z)\rangle \text{ and } |\psi_i\rangle = \frac{1}{2^{m/2}} \sum_{z=0}^{2^m-1} |z\rangle\,|h(z)\rangle \ ,$$

$$\text{then:} \quad \| \,|\psi_e\rangle\, \|^2 := \| \,|\psi_r\rangle - |\psi_i\rangle\, \|^2 = (p2^m)\frac{4}{2^m} = 4p \ , \tag{8}$$

and, since the last QFT step is a unitary, it preserves the Euclidean distance:

$$\|(I \otimes QFT)(|\psi_r\rangle) - (I \otimes QFT)(|\psi_i\rangle)\| = \| \,|\psi_r\rangle - |\psi_i\rangle\, \| = 2\sqrt{p} \ . \tag{9}$$

Since we perform μ measurements in total, we have μ copies of the subroutine, and we can bound the Euclidean distance between the real and ideal runs as follows:

$$\|(|\psi_r\rangle)^{\otimes\mu} - (|\psi_i\rangle)^{\otimes\mu}\| = \sqrt{\mu}(2\sqrt{p}) \ . \tag{10}$$

Therefore, the total variation distance between the probability distributions resulting from measuring the ideal and the real runs is bounded by: $4(2\sqrt{\mu}\sqrt{p})$ by Lemma 3.6 in [7]. The result of the "ideal" and "real" algorithm is a function of these measurement results. Therefore:

$$|p_r - p_i| \leq 4\sqrt{\mu}2\sqrt{p} \implies p_r \geq p_i - 8\sqrt{\mu}\sqrt{p} \ . \tag{11}$$

This concludes the proof. □

Table 1. Quantum space-optimized factoring algorithms. The input is an n-bit number. The gate count is for the entire algorithm, as ours (last lines) requires $\mathcal{O}(\log n)$ independent runs. The difference between RSA and general integers comes from the variant of Shor's algorithm and post-processing that can be applied (see Section 5 for the general case).

Algorithm	Qubits	Gates (Toffolis)
[3,36,22,16,15]	$2n + \mathcal{O}(1)$	$\mathcal{O}\left(n^3 \log n\right)$
[38]	$1.5n + \mathcal{O}(1)$	$\mathcal{O}\left(n^3 \log n\right)$
[19]	$3n + 0.002n \log n$	$0.3n^3 + 0.0005n^3 \log n$
[24]	$2n + \mathcal{O}(\log n)$	$\mathcal{O}\left(n^{2.29}\right)$
This work (RSA)	$\frac{n}{2} + o(n)$	$\mathcal{O}\left(n^3 \log n\right)$
This work (general integers)	$n + o(n)$	$\mathcal{O}\left(n^3 \log n\right)$

3.5 Overview of Related Works for Factoring

Since Shor's original paper, many authors have optimized the algorithm in terms of depth, gate count or qubit usage. Since we focus here on space requirements, we list in Table 1 the number of qubits required by different logical circuit implementations.

The complexity of implementing Shor's algorithm depends crucially on the multiplication circuit that is used. Since multiplication of n-bit integers can be done in $\mathcal{O}(n \log n)$ bit operations [23], there exists a quantum circuit for multiplying n-bit integers using $\mathcal{O}(n \log n)$ gates and qubits, and some of the gate counts given in the literature follow this asymptotic rule. However, when aiming for the space complexity, the multiplication circuits used have been mostly schoolbook multiplication (combined with advanced techniques such as windowed arithmetic [18] or coset representation of modular integers [38]) or QFT-based [9], requiring at least $\mathcal{O}\left(n^2\right)$ gates. Recently, Kahanamoku-Meyer and Yao proposed a new efficient multiplication circuit requiring few ancillas [24], which performs better asymptotically and in practice.

Regev [31] introduced another quantum factoring algorithm which can be regarded as a multi-dimensional variant of Shor's, allowing to reduce the circuit size by a factor $\mathcal{O}(\sqrt{n})$ at the expense of making $\mathcal{O}(\sqrt{n})$ measurements. If schoolbook multiplication is used, this reduces the gate count of each run from $\mathcal{O}\left(n^3\right)$ to $\mathcal{O}\left(n^{5/2}\right)$. Ragavan and Vaikuntanathan [30] showed how to implement this algorithm with $\mathcal{O}(n)$ qubits. The algorithm was also extended to Discrete Logarithms in $\mathbb{Z}_P^*$ [14]. However, regarding space complexity, Regev's algorithm is worse than previous works, and [30] reports $12.32(1 + \varepsilon)n$ qubits (where ε is a small heuristic constant). Whether our strategy can be applied here remains an open question.

4 Compressed Modular Multi-Product Circuit

In this section, we explain the core of our approach: a space-efficient circuit for computing a *compressed modular multi-product*.

We work in the group $\mathbb{Z}_N^*$ and denote $n = \lceil \log_2 N \rceil$. Let r be a constant which will be chosen appropriately later on. A modular exponentiation circuit as used in Shor's and Ekerå-Håstad's algorithms reduces to a *modular multi-product*, as explained in Section 3.1.

Let $m \leq 2n$ and $A_0, \ldots, A_{m-1}$ be arbitrary n-bit integers. Given an input X identified as a bit-string $x_0, \ldots, x_{m-1}$, we define: $A_X := \prod_{i=0}^{m-1} A_i^{x_i}$. Our circuit will compute $[[A_X]_N]_{2^r}$ using $\mathcal{O}\big(nm^2\big)$ gates and $\mathcal{O}\Big(\log n + \frac{m}{\log n}\Big) = o(n)$ ancilla space, for a total depth $\mathcal{O}\big(nm \log^3 n\big)$ (Theorem 2). This space can even be reduced to $\mathcal{O}(\log n)$, but the depth would increase to $\mathcal{O}\big(nm^2\big)$.

This circuit *fails on some inputs*. However, under Heuristic 2 (detailed below), its probability of failure can be reduced to an arbitrary constant.

4.1 Main Idea: a Residue Number System (RNS)

We will use an RNS to represent the multi-product A_X, i.e., a number of at most nm bits. Let $\mathcal{P} = \{p_1, \ldots, p_\ell\}$ be the prime numbers for the RNS. By Lemma 1 we have $\ell = \mathcal{O}(mn/\log(mn)) = \mathcal{O}(mn/\log n)$ and $\forall p \in \mathcal{P}, p \leq mn$. We let $w := \lceil \log_2 \max_{p\in\mathcal{P}} p \rceil$. The product $M := \prod_{p\in\mathcal{P}} p$ is upper bounded by: $M < 2^{nm} p_\ell \leq 2^{nm+w}$. Indeed, by definition of the RNS, we have $\prod_{i=0}^{\ell-1} p_i < 2^{nm} \implies M < 2^{nm} p_\ell$. Thus M can get bigger than 2^{nm} due to the last prime factor, but only by a small number of bits. Finally, we define: $\alpha := \left\lceil \log_2 \left(\sum_{p\in\mathcal{P}} p \right) \right\rceil$. In particular $\alpha \leq w + \lceil \log_2 \ell \rceil$.

For all p, let $M_p = M/p$ (which is an integer) and $w_p = (M_p)^{-1} \bmod p$, and notice that $M_p w_p < M$. Let us define:

$$A'_X := \sum_{p\in\mathcal{P}} \underbrace{\Big[A_X\Big]_p}_{w \text{ bits}} \underbrace{M_p w_p}_{mn+w \text{ bits}} \leq 2^{mn+w+\alpha} \ . \tag{12}$$

From there, we need to perform two layers of modular reduction: modulo M and modulo N, as we have: $[A_X]_N = [[A'_X]_M]_N$. We define two quantities which are the quotients of both Euclidean divisions:

$$\begin{cases} q_M := \lfloor A'_X / M \rfloor = \left\lfloor \frac{\sum_{p\in\mathcal{P}} [A_X]_p M_p w_p}{M} \right\rfloor \\ Q_N := \lfloor [A'_X]_M / N \rfloor = \left\lfloor \frac{\left(\sum_{p\in\mathcal{P}} [A_X]_p M_p w_p\right) - q_M M}{N} \right\rfloor \end{cases} \tag{13}$$

We notice here that q_M is a small number, which is why we used a lowercase letter for the notation. Indeed, we have: $A'_X \leq 2^{mn+w+\alpha}$ and $M \geq 2^{nm}$ by definition of the RNS, thus $q_M < 2^{w+\alpha} = \mathsf{poly}(mn)$.

Finally, we can express our end result as:

$$\begin{aligned} [[[A'_X]_M]_N]_{2^r} &= [(A'_X - q_M M) - Q_N N]_{2^r} \\ &= [A'_X]_{2^r} - [q_M]_{2^r} [M]_{2^r} - [Q_N]_{2^r} [N]_{2^r} \bmod 2^r \ . \end{aligned}$$

We observe that $[A'_X]_{2^r}$ can be computed as:

$$[A'_X]_{2^r} = \sum_p [[A_X]_p]_{2^r} [M_p w_p]_{2^r} \bmod 2^r \ ,$$

using a series of multiplications and additions modulo 2^r. This cost is insignificant as r is a small constant ($r = 22$ will be used in Section 5). Besides, $[M]_{2^r}$ and $[N]_{2^r}$ are constants.

It can be also noticed that in this expression, $[A'_X]_{2^r} - [q_M]_{2^r} [M]_{2^r}$ is equal to zero with large probability. This is because A_X is a multi-product of random integers, half of which are multiples of 2. As r is typically much smaller than $m/2$, we have that $[A_X]$ is often divisible by 2^r, thus $[A_X]_{2^r}$ is zero.

Next, we will see how we compute q_M, then $[q_M]_{2^r}$ and $[Q_N]_{2^r}$. For this, we will make two successive approximations. The first one requires X to have bounded Hamming weight, which is ensured with high probability if X is selected uniformly at random. The second one is probabilistic, and its probability of success is a tunable parameter in our algorithm.

The Hamming Weight Constraint on X. The success of our algorithm depends on the Hamming weight of X. We have the following.

Lemma 4. *When $X \in \{0,1\}^m$ is chosen u.a.r., the following holds with probability $1 - 2\exp\left(-\frac{4}{6}m^{1/3}\right)$:*

$$hw(X) \leq \frac{m}{2} + m^{2/3} \qquad \text{and} \qquad A_X \leq N^{\frac{m}{2}+m^{2/3}} \ . \tag{14}$$

Proof. Since we just need to count the 1-coordinates of X, we use a multiplicative Chernoff bound:

$$\Pr\left(\left|hw(X) - \frac{m}{2}\right| \geq 2m^{-1/3} \times \frac{m}{2}\right) \leq 2\exp\left(-(2m^{-1/3})^2 \frac{m}{6}\right)$$
$$\implies \Pr\left(\left|hw(X) - \frac{m}{2}\right| \geq m^{2/3}\right) \leq 2\exp\left(-\frac{4}{6}m^{1/3}\right) \ .$$

The second inequality is a direct implication. □

An immediate consequence of this fact is that we do not need our RNS to represent nm-bit numbers, but only $n(\frac{m}{2} + m^{2/3})$-bit numbers. To be more precise, we modify our definition as follows: we let $\mathcal{P} = \{p_1, \ldots, p_\ell\}$ be $\mathcal{O}(mn/\log n)$ prime numbers such that $M = \prod_{p\in\mathcal{P}} p \geq 2^{n(\frac{m}{2}+2m^{2/3})}$. The additional factor $2^{nm^{2/3}}$ has a reason which will be explained below.

4.2 Reduction Modulo M and First Approximation

We will now see how we compute q_M. Recall its definition:

$$q_M := \left\lfloor \frac{\sum_{p\in\mathcal{P}} [A_X]_p w_p M_p}{M} \right\rfloor = \left\lfloor \sum_{p\in\mathcal{P}} [A_X]_p \frac{w_p}{p} \right\rfloor \ . \tag{15}$$

Here p, w_p, $[A_X]_p$ and eventually q_M are all small numbers (bit-size logarithmic in n), so intuitively we may compute this sum directly by approximating $\frac{1}{p}$ by $\frac{1}{2^u}\lfloor 2^u/p \rfloor$, where $u = \mathcal{O}(\log n)$ is chosen appropriately.

We note that the following Lemma is essentially a variant of the Explicit CRT, where we also allow a small probability of failure, which is asymptotically negligible in n. This result in itself is not new, see for example [5,6]. It would be possible here to use the previous Explicit CRT from [5,6], and it would just induce a small change in the formula for q_M.

Lemma 5. *Let $u = \left\lceil \log_2 \sum_{p\in\mathcal{P}} p^2 \right\rceil$. Asymptotically in n, with probability $1 - negl(n)$, the following holds:*

$$q_M = \left\lfloor \frac{1}{2^u} \left(\sum_{p\in\mathcal{P}} [A_X]_p w_p \lfloor 2^u/p \rfloor \right) \right\rfloor + 1 \ . \tag{16}$$

Proof. Recall that we chose our RNS product M such that $M \geq 2^{n(\frac{m}{2}+2m^{2/3})}$, but with high probability we have $A_X \leq 2^{n(\frac{m}{2}+m^{2/3})}$. As a consequence: $0 < \frac{A_X}{M} \leq 2^{-nm^{2/3}}$. and:

$$\left\{ \sum_{p\in\mathcal{P}} [A_X]_p \frac{w_p}{p} \right\} = \frac{A_X}{M} \leq 2^{-nm^{2/3}} \ .$$

By choosing $u = \left\lceil \log_2 \sum_{p\in\mathcal{P}} p^2 \right\rceil$ as in the statement of the lemma, we have $2^u > \sum_{p\in\mathcal{P}} p^2$. Then we have the following inequalities (for $p \neq 2$):

$$\frac{2^u}{p} \geq \left\lfloor \frac{2^u}{p} \right\rfloor + \frac{1}{p} \text{ and } \left\lfloor \frac{2^u}{p} \right\rfloor \geq \frac{2^u}{p} - 1 \ . \tag{17}$$

On the one hand, we have:

$$\left(\sum_{p\in\mathcal{P}} [A_X]_p \frac{w_p}{p} \right) - \left(\sum_{p\in\mathcal{P}} [A_X]_p \frac{w_p}{2^u} \right) \leq \frac{1}{2^u} \left(\sum_{p\in\mathcal{P}} [A_X]_p w_p \lfloor 2^u/p \rfloor \right) \ . \tag{18}$$

Therefore, our choice of u ensures that:

$$q_M - 1 \geq \left\lfloor \frac{1}{2^u} \left(\sum_{p\in\mathcal{P}} [A_X]_p w_p \lfloor 2^u/p \rfloor \right) \right\rfloor \ . \tag{19}$$

On the other hand:

$$\frac{1}{2^u} \left(\sum_{p\in\mathcal{P}} [A_X]_p w_p \lfloor 2^u/p \rfloor \right) \leq \left(\sum_{p\in\mathcal{P}} [A_X]_p \frac{w_p}{p} \right) \left(1 - 2^{-u}\right) \ . \tag{20}$$

Asymptotically in n, we have seen that the fractional part of $\sum_{p\in\mathcal{P}}[A_X]_p\frac{w_p}{p}$ is very small. With the value of u chosen, we have $u = \mathsf{polylog}(n)$, and so:

$$\left\{\sum_{p\in\mathcal{P}}[A_X]_p\frac{w_p}{p}\right\} < 2^{-u}\left(\sum_{p\in\mathcal{P}}[A_X]_p\frac{w_p}{p}\right) \quad . \tag{21}$$

Indeed, $w_p \geq 1$ for all p, and for at least one $p \in P$ we have $[A_X]_p \geq 1$ (otherwise A_X would be divisible by all primes, which is impossible since it is smaller than M). This means that: $\sum_{p\in\mathcal{P}}[A_X]_p\frac{w_p}{p} \geq 1/\mathcal{O}(mn)$, due to the bound on the primes. Furthermore by choosing $u = \mathsf{polylog}(n)$ we have (asymptotically) $2^{-u}/\mathcal{O}(mn) < 2^{-nm^{2/3}}$, ensuring that the result holds.

Consequently this inequality entails:

$$\frac{1}{2^u}\left(\sum_{p\in\mathcal{P}}[A_X]_p w_p \left\lfloor 2^u/p\right\rfloor\right) < \sum_{p\in\mathcal{P}}[A_X]_p\frac{w_p}{p} - \left\{\sum_{p\in\mathcal{P}}[A_X]_p\frac{w_p}{p}\right\} = q_M \quad , \tag{22}$$

which completes our proof. □

At this point, we notice that q_M can be computed on the fly (by computing the $[A_X]_p$) using low space. The value of q_M is important both for having $[q_M]_{2^r}$, but also because it appears in Q_N. We can now turn ourselves towards the second step, i.e., the computation of $[Q_N]_{2^r}$.

Remark 1. Strictly speaking, this first approximation is not necessary. We could choose an RNS representing $n(m+1)$-bit numbers, and the computation of q_M by Equation 16 would then work with certainty for all inputs X. However, this source of errors is negligible, and the gain in practice is significant enough to justify taking a smaller RNS.

4.3 Reduction Modulo N and Second Approximation

For the computation of $[Q_N]_{2^r}$, we will use Barrett reduction. We introduce a parameter t_N to satisfy the condition of Proposition 1, i.e., $2^{t_N} \geq A_X N$, so we can choose $t_N = \left\lceil n(\frac{m}{2} + m^{2/3})\right\rceil + n$. We also remark that:

Lemma 6. *For any choice of X, A_X is not a multiple of N.*

This follows from the fact that all numbers in the multi-product are powers of A, which is (assumed to be) prime with N. Therefore, Proposition 1 applies and Q_N has the expression:

$$Q_N = \left\lfloor \frac{1}{2^{t_N}}\left(\sum_{p\in\mathcal{P}}[A_X]_p M_p w_p \left\lfloor 2^{t_N}/N\right\rfloor - q_M M\left\lfloor 2^{t_N}/N\right\rfloor\right)\right\rfloor \quad . \tag{23}$$

We have:

$$[Q_N]_{2^r} = \left(\sum_{p \in \mathcal{P}} [A_X]_p M_p w_p \left\lfloor 2^{t_N}/N \right\rfloor - q_M M \left\lfloor 2^{t_N}/N \right\rfloor \right) \gg t_N \bmod 2^r \ , \quad (24)$$

where $\gg$ is a bit-shift, i.e., we need to compute the bits at positions $t_N + 1$ to $t_N + r$ in the previous equation. We will use the fact that q_M is known to us at this point. For convenience, we replace the difference by a sum, since we work modulo 2^{t_N+r}:

$$[Q_N]_{2^r} = \left(\sum_{p \in \mathcal{P}} [A_X]_p M_p w_p \left\lfloor 2^{t_N}/N \right\rfloor + q_M (2^{t_N+r} - M \left\lfloor 2^{t_N}/N \right\rfloor) \right) \gg t_N \bmod 2^r \ .$$

Intuitively, there is no need to compute the entire sum, as the least significant bits only have a very small influence on the bits from $t_N + 1$ to $t_N + r$. To view this, we first decompose the modular residues $[A_X]_p$ and q_M on individual bits, and express $[Q_N]_{2^r}$ as a large sum:

$$[Q_N]_{2^r} = \left(\sum_{p \in \mathcal{P}} \sum_i ([A_X]_p)_i \, 2^i M_p w_p \left\lfloor 2^{t_N}/N \right\rfloor \right.$$
$$\left. + \sum_i (q_M)_i 2^i \left(2^{t_N+r} - M \left\lfloor 2^{t_N}/N \right\rfloor \right) \right) \gg t_N \bmod 2^r \ .$$

Therefore we can see $[Q_N]_{2^r}$ as a sum of $\leq \sum_{p \in \mathcal{P}} \lceil \log_2 p \rceil + \lceil \log_2 q_M \rceil$ integers. How many depends on the current values of $[A_X]_p$ and q_M (but we prefer to overestimate). These integers behave as if they were drawn uniformly at random[3].

We introduce a parameter u' and approximate $[Q_N]_{2^r}$ by truncating these integers to u' bits, and taking the bits $u' + 1$ to $u' + r$ in the sum:

$$[Q_N]_{2^r} \simeq \left(\sum_{p \in \mathcal{P}} \sum_i ([A_X]_p)_i \left\lfloor 2^i M_p w_p \left\lfloor 2^{t_N}/N \right\rfloor / 2^{t_N - u'} \right\rfloor \right.$$
$$\left. + \sum_i (q_M)_i \left\lfloor 2^i \left(2^{t_N+r} - M \left\lfloor 2^{t_N}/N \right\rfloor \right) / 2^{t_N-u'} \right\rfloor \right) \gg u' \bmod 2^r \ . \quad (25)$$

The amount of precision we need is related to the number of integers we are summing. In a sum of t random integers we need a precision of at least $\lceil \log_2 t \rceil$ bits to succeed with constant probability. The reason is the same as in *oblivious carry runways* [17]. When truncating with $\lceil \log_2 t \rceil + \nu$ bits, the probability that carries propagate all the way from the truncated part of the integers to the wanted bit (and the bit is flipped as a result) is $2^{-\nu}$.

[3] This is not exactly the case because they have some zero LSBs, but since we are looking at the bits $t_N + 1$ to $t_N + r$, which are quite far in the sum, the LSBs are insignificant.

This is what happens if the integers are drawn at random. This is not the case for us, as these integers are precomputed cofactors depending on the RNS. Experimentally, we observe that this property remains satisfied, and formulate it as a heuristic.

Heuristic 3. When choosing $u' = \left\lceil \log_2(\sum_{p\in\mathcal{P}} \lceil \log_2 p \rceil + \lceil \log_2 q_M \rceil) \right\rceil + \nu$, for random inputs X, Equation 25 is an equality with probability $\geq 1 - 2^{-\nu}$.

In fact, this heuristic is the only one underlying Heuristic 2. Indeed, all errors come from the computation of q_M and $[Q_N]_{2^r}$. The first approximation is not heuristic: the X in A_X is required to have bounded Hamming weight, which happens with high probability in our case, according to Lemma 4. If Heuristic 3 is true, then Heuristic 2 is also true: the probability of error can be brought down an arbitrary constant, at a low cost.

4.4 Summary

Thanks to the approximations, both q_M and $[Q_N]_{2^r}$ are computed with $o(n)$ space. More precisely, let $\gg$ be a bitwise right-shift, then we have:

$$\begin{cases} q_M = \left(\left(\sum_{p\in\mathcal{P}} [A_X]_p \underbrace{w_p \left\lfloor 2^u/p \right\rfloor}_{\text{Precomputed}} \right) \gg u \right) + 1 \\ [Q_N]_{2^r} = \Bigg(\sum_{p\in\mathcal{P}} \sum_i \left([A_X]_p\right)_i \underbrace{\left[\left\lfloor 2^i M_p w_p \left\lfloor 2^{t_N}/N \right\rfloor / 2^{t_N-u'} \right\rfloor \right]_{2^{u'+r}}}_{\leq 2^{u'+r} \text{ and precomputed}} \\ \qquad + \sum_i (q_M)_i \underbrace{\left[\left\lfloor 2^i \left(2^{t_N+r} - M \left\lfloor 2^{t_N}/N \right\rfloor\right) / 2^{t_N-u'} \right\rfloor \right]_{2^{u'+r}}}_{\leq 2^{u'+r} \text{ and precomputed}} \Bigg) \gg u' \end{cases} \tag{26}$$

Therefore, we only need to compute large (controlled) sums of precomputed integers (which we call *cofactors*), depending on the bits of $[A_X]_p$ and q_M. Since r is a constant, $u = \mathcal{O}(\log(mn))$, $q_M = \mathsf{poly}(mn)$ and $u' = \mathcal{O}(\log(mn))$, the amount of temporary storage for q_M and Q_N is in $\mathcal{O}(\log(mn)) = \mathcal{O}(\log n)$. The cofactors are the following:

$$\begin{cases} B_p := [M_p w_p]_{2^r} \\ C_{p,i} := \left[2^i w_p \left\lfloor 2^u/p \right\rfloor \right]_{2^{u+\lceil \log_2 q_M \rceil + 1}} \\ D_{0,i} := \left[\left\lfloor 2^i \left(2^{t_N+r} - M \left\lfloor 2^{t_N}/N \right\rfloor\right) / 2^{t_N-u'} \right\rfloor \right]_{2^{u'+r}} \\ D_{p,i} := \left[\left\lfloor 2^i M_p w_p \left\lfloor 2^{t_N}/N \right\rfloor / 2^{t_N-u'} \right\rfloor \right]_{2^{u'+r}} \end{cases} \tag{27}$$

We summarize the resulting algorithm as Algorithm 1, with a layout close to how it will be implemented as a quantum circuit.

Under the heuristics above, we can prove our main result.

Algorithm 1 Approximate multi-product algorithm.

Input: X
Output: $[[A_X]_N]_{2^r}$
Precomputed: cofactors $B_p, C_{p,i}, D_{0,i}, D_{p,i}$

$\mathsf{qM} \leftarrow 0$ ▷ value of q_M ($\mathcal{O}(\log n)$ bits)
$\mathsf{QN} \leftarrow 0$ ▷ value of $[Q_N]_{2^r}$ ($\mathcal{O}(\log n)$ bits)
$\mathsf{Res} \leftarrow 0$ ▷ value of the result (r bits)
for all p in the RNS **do**
  $x \leftarrow [A_X]_p$ ▷ using the subroutine
  $x := x_0 + 2x_1 + ...$
  for all $0 \leq i \leq \lceil \log_2 p \rceil$ **do**
    if $x_i = 1$ **then**
      $\mathsf{qM} \leftarrow \mathsf{qM} + C_{p,i}$ ▷ The computation is done on $u + \lceil \log_2 q_M \rceil + 1$ bits
      $\mathsf{QN} \leftarrow \mathsf{QN} + D_{p,i}$ ▷ The computation is done on $u' + r$ bits
    end if
  end for
  $\mathsf{Res} \leftarrow \mathsf{Res} + [x]_{2^r} B_p$ ▷ Modulo 2^r
end for
$\mathsf{qM} \leftarrow (\mathsf{qM} \gg u) + 1$
$\mathsf{Res} \leftarrow \mathsf{Res} - [\mathsf{qM}]_{2^r}[M]_{2^r}$
$q_M := b_0 + 2b_1 + \dots$
for all $0 \leq i \leq \lceil \log_2 q_M \rceil$ **do**
  If $b_i = 1$ **then** $\mathsf{QN} \leftarrow \mathsf{QN} + D_{0,i}$
end for
$\mathsf{QN} \leftarrow \mathsf{QN} \gg u'$
$\mathsf{Res} \leftarrow \mathsf{Res} - [\mathsf{QN}]_{2^r}[N]_{2^r}$
Return Res

Theorem 2. *Assume that $m \leq 2n$ and r is a constant. For any constant $\varepsilon > 0$, there exists a reversible logical circuit for compressed multi-product which, on input $(X, 0)$ (where X is represented on a register of size m, and 0 is an ancilla register), returns $(X, [[A_X]_N]_{2^r})$, and succeeds with probability $1 - \varepsilon$ for inputs X chosen uniformly at random, under Heuristic 3. It uses $\mathcal{O}\big(nm^2\big)$ gates, depth $\mathcal{O}\big(nm \log^3 n\big)$ and $\mathcal{O}\Big(\log n + \frac{m}{\log n}\Big)$ ancillas.*

Proof. The circuit layout follows Algorithm 1. We set the parameters u and u' so that the probability of success of the circuit is sufficient.

Throughout the computation we maintain a register for q_M, a register for the result (of r bits) and a register for Q_N. Each time a new residue is computed, we perform $\mathcal{O}(\log n)$ controlled additions by constants in the registers for q_M and Q_N. We also sum $[[A_X]_p]_{2^r}[M_p w_p]_{2^r}$ in the result register.

By Lemma 1, there are $\mathcal{O}((nm)/\log n)$ primes and we compute each residue only once. For each residue, we use the multi-product circuit of Lemma 8, which requires $\frac{m}{\log n}$ ancilla qubits to reach a low depth.

Once the whole sum for q_M has been computed, we shift it by u bits and increment. We have obtained q_M. At this point the result register also contains $[A'_X]_{2^r}$, and after subtracting $[\mathsf{qM}]_{2^r}[M]_{2^r}$ it contains $[A'_X - q_M M]_{2^r}$.

We use the bits of q_M to complete the sum for Q_N, by making new controlled additions for each one with their own precomputed cofactors. This step is computationally negligible with respect to the other additions that we just performed.

When we have finished the sum for Q_N, we select its bits at position $u'+1$ to $u'+r$, which give $[Q_N]_{2^r}$. We subtract $[Q_N]_{2^r}[N]_{2^r}$ from the result register, which gives the result of the algorithm.

Then, we erase the accumulator registers by recomputing the residues and performing a series of controlled subtractions or additions.

Since the accumulator registers are also of size $\mathcal{O}(\log n)$, the circuit uses $\mathcal{O}(\log n)$ working bits in addition of those required for the RNS residues. The gate count is dominated by the computation of the residues, which we do using Lemma 8. □

4.5 Modular Multi-Product in the RNS

The main problem with the computation of $[A_X]_p$ is its sequentiality. Indeed, if we follow the blueprint of modular multi-product in Shor's algorithm, we do a sequence of m controlled modular multiplications by $[A_i]_p$. This takes depth $\widetilde{\mathcal{O}}(m)$, and gates are applied only on a work register of size $\mathcal{O}(\log p) = \mathcal{O}(\log n)$, while the control qubits remain idle for the most part.

In the following, since we work at low scales, the asymptotic formulas that we give assume the use of schoolbook addition and multiplication circuits: on $\log n$ bits, addition costs $\mathcal{O}(\log n)$ gates and depth, and multiplication costs $\mathcal{O}\big(\log^2 n\big)$ gates and depth.

Our first improvement on the naive multi-product is to replace the controlled modular multiplications by a sequence of controlled additions.

Lemma 7. *Let $a_0, \ldots, a_{m-1} \in \mathbb{Z}_p$, and p be a prime, with $m \leq 2n$ and $p = \mathcal{O}(mn)$. There exists a circuit performing the multi-product modulo p:*

$$|x_0, \ldots, x_{m-1}\rangle\,|0\rangle \mapsto |x_0, \ldots, x_{m-1}\rangle\,|[\prod_i a_i^{x_i}]_p\rangle \quad ,$$

using $\mathcal{O}(m \log n)$ gates, $\mathcal{O}(m \log n)$ depth and $\mathcal{O}(\log n)$ ancilla qubits.

Proof. First of all, we set aside the a_i which are zero (modulo p). If one of the corresponding bits x_i is 1, then the result is 0. The remainder of this proof considers for simplicity that all the a_i are nonzero.

We choose a generator g of $\mathbb{Z}_p^*$ and precompute the discrete logarithms of the a_i modulo p. Let $\alpha_i \leq p-1$ be such that: $a_i \equiv g^{\alpha_i} \bmod p$. Then we have:

$$\prod_i a_i^{x_i} \equiv g^{\sum_{i=0}^{m-1} x_i \alpha_i} \bmod p \ . \tag{28}$$

We first compute $\sum_{i=0}^{m-1} x_i\alpha_i$, which is smaller than $m(p-1) = \mathcal{O}\big(m^2 n\big)$. Then, we use a $\mathcal{O}(\log n)$-bit exponentiation circuit modulo p: we precompute the $g^{2^i} \mod p$, we read off the $\mathcal{O}(\log n)$ bits of $\sum_{i=0}^{m-1} x_i\alpha_i$, and perform $\mathcal{O}(\log n)$ modular multiplications in gate count: $\mathcal{O}\big(\log^3 n\big)$. Only $\mathcal{O}(\log n)$ ancilla space is needed for these two operations. □

Next, we can trade depth for memory during the computation of the sum.

Lemma 8. *There exists a quantum circuit for the multi-product modulo p that uses $\mathcal{O}(m \log n)$ gates, $\mathcal{O}\Big(\frac{m}{\log n}\Big)$ ancilla qubits and depth $\mathcal{O}\big(\log^4 n\big)$.*

Proof. We change the way we compute $\sum_{i=0}^{m-1} x_i\alpha_i$, as follows. We select a parameter $2 < k \leq m$ and separate the numbers into k groups. We compute the sum of each group using an addition tree. As there are (m/k) numbers to add, the tree has depth $\log_2(m/k)$. We need $\mathcal{O}\big(\frac{m}{k} \log n\big)$ ancilla qubits to write all its nodes (starting from the first numbers), and $\mathcal{O}\big(\frac{m}{k} \log n\big)$ gates to compute all of them. The result of each smaller sum is added into an accumulator register, which contains the result of the whole sum. Since there are k groups, the total depth is: $\mathcal{O}(k \times \log(m/k) \times \log n) = \mathcal{O}\big(k(\log n)^2\big)$ and the total gate count is: $\mathcal{O}(m \log n)$. By selecting $k = (\log n)^2$, we achieve a space in $\mathcal{O}\Big(\frac{m}{\log n}\Big)$ and a depth $\mathcal{O}\big((\log n)^4\big)$ (which becomes dominating), for the same gate count. □

We obtained better results with the following optimization, even though it increases slightly the asymptotic gate count.

Lemma 9. *There exists a circuit for the multi-product mod p using $\mathcal{O}\big(m \log^2 n\big)$ gates, $\mathcal{O}\Big(m\frac{\log\log n}{\log n}\Big) = o(m)$ ancilla qubits and depth $\mathcal{O}\big(\log^3 n\big)$.*

Proof. We write the binary representation of α_i: $\forall i, \alpha_i = \sum_k (\alpha_i)_k 2^k$. Then:

$$\sum_{i=0}^{m-1} x_i\alpha_i = \sum_k \left(\sum_{i=0}^{m-1} x_i (\alpha_i)_k \right) 2^k \ . \tag{29}$$

A strategy to compute the sum in $\mathcal{O}(\log n)$ steps follows: we initialize a $\mathcal{O}(\log n)$-qubit accumulator. At each step, we compute $\sum_{i=0}^{m-1} (\alpha_i)_k$, shift the value by k bits and add it to our accumulator.

We now focus on the computation of $\sum_{i=0}^{m-1} x_i (\alpha_i)_k$ for a fixed k. Since $(\alpha_i)_k$ are precomputed values, this is equivalent to computing $\sum_{i \in I} x_i$ for a fixed subset $I \subseteq \{0, \ldots, m-1\}$. To do this with a minimal use of ancillas, we adopt the following strategy.

We separate the bits of I (less than m) into groups of size $k = \mathcal{O}(\log n)$. To each of these groups, we append an ancillary register of $\mathcal{O}(\log\log n)$ qubits. We use a sequence of $\log n$ in-place incrementor circuits to sum the bits into this register one by one. Then, we sum these ancillary registers two by two. Each

time we sum two registers, we use an addition circuit and append two more qubits for the inevitable carries. The total ancilla space is therefore:

$$\frac{m}{k}\log k + \frac{m}{k}\left(1+\frac{1}{2}+\ldots\right) = \mathcal{O}\left(m\frac{\log\log n}{\log n}\right) \quad .$$

The first step (local and sequential) uses $\frac{m}{\log n} \times \log n \times \log n = \mathcal{O}(m \log n)$ gates and depth $\mathcal{O}\big(\log^2 n\big)$. The second step (global and parallel) needs depth $\mathcal{O}\big(\log \frac{m}{k} \times \log n\big) = \mathcal{O}\big(\log^2 n\big)$ as well under the same assumption, and uses $\mathcal{O}(m)$ gates.

Finally, we need to do this $\mathcal{O}(\log n)$ times. The gate count increases to $\mathcal{O}\big(m \log^2 n\big)$ and the total depth is $\mathcal{O}\big(\log^3 n\big)$, which is the same as the modular exponentiation circuit. □

5 Space-efficient Quantum Factoring and Discrete Logarithms

In this section, we put together the three ingredients of our algorithm: 1. the Ekerå-Håstad algorithm; 2. our new compression result; 3. the compressed multi-product circuit. This gives us an "explicit" compressed Ekerå-Håstad algorithm (Algorithm 2). We prove its correctness under heuristics (Theorem 2), first for factoring, and then we extend to short discrete logarithms (Corollary 1).

5.1 New Compressed Subroutine

An important ingredient in our analysis is that we do not directly compress the multi-product output to one bit. In order to rely only on Heuristic 1 (the modular multi-product is roughly a random periodic function), we truncate the multi-product to r bits first, and then we take the dot-product with randomly chosen masks β. As the mask is chosen before running the quantum algorithm, this only modifies a few gates in the specification of the circuit, and this has negligible impact on its computational resources[4].

The analysis of the Ekerå-Håstad algorithm [15] relies on the probability to measure a "good" output y. We show that this probability is only halved, and that half of the measurement results are 0 (which is discarded). This also tells that the size of Y at Step 12 is essentially the same as before, and that the post-processing routine runs unchanged. The proof of Theorem 3 (deferred to Section B) relies on concentration inequalities, leveraging the fact that we have many possible choices of β, and so the deviation of a probability of measurement from its average can be bounded.

Theorem 3. *Under Heuristic 1, with probability 0.99 over the choice of N, at Step 8 in Algorithm 2:*

- *the probability to measure 0 is at most $\frac{1}{2} + 0.01$ plus half of the probability to measure 0 in the uncompressed Ekerå-Håstad subroutine*
- *the probability to measure a "good" y is at least $\frac{1}{16} - 0.01$*

[4] In fact, we conjecture that it should be possible to get rid entirely of this mask β.

Algorithm 2 New space-efficient quantum factoring for RSA integers. In the DL case, A is an input of the algorithm and it stops when D is output.

Input: N
Output: P, Q st. $P \times Q = N$
Parameters: $n = \lceil \log_2 N \rceil$, s, μ, $m = \frac{n}{2} + 2\left\lceil \frac{n}{2s} \right\rceil$, $r = 22$

▷ Classical steps

1: Select a random number $G < N$ invertible modulo N
2: Let $A = G^{(N-1)/2-2^{n/2-1}}$
3: Precompute G^{2^i} for $0 \leq i < \frac{n}{2} + \left\lceil \frac{n}{2s} \right\rceil$ and A^{2^j} for $0 \leq j < \left\lceil \frac{n}{2s} \right\rceil$
4: Setup Algorithm 1 so that its probability of error p satisfies: $8\sqrt{4\mu}\sqrt{p} < 0.08$
5: Precompute the cofactors required by Algorithm 1 and define the multi-product circuit for $A_0, \ldots, A_{m-1} = G^{2^0}, \ldots, G^{2^{\left(\frac{n}{2}+\left\lceil \frac{n}{2s} \right\rceil - 1\right)}}, A^{-2^0}, \ldots, A^{-2^{\left(\left\lceil \frac{n}{2s} \right\rceil - 1\right)}}$

▷ Quantum subroutines

$Y \leftarrow \emptyset$
6: **while** $|Y| < \mu$ **do**
7: Select a random non-zero mask β of r bits
8: Run the "compressed Ekerå-Håstad" sub-routine with the following compression of the multi-product circuit:

$$(x_0, \ldots, x_{m-1}) \mapsto \beta \cdot \left(\prod A_i^{x_i} \bmod N \bmod 2^r \right)$$

9: Measure y
10: **If** $y \neq 0$, $Y \leftarrow Y \cup \{y\}$
11: **end while**

▷ Classical post-processing

12: Run the Ekerå-Håstad post-processing routine on Y
13: Let D be the discrete logarithm of A. From $D = \frac{P-1}{2} + \frac{Q-1}{2} - 2^{n/2-1}$, deduce $P + Q$.
14: Deduce P and Q
15: **Return** P, Q

5.2 Complete Algorithm

We can now prove our result for factoring.

Theorem 4. *Under Heuristic 1 and Heuristic 2, if the Ekerå-Håstad algorithm running with parameters s and μ succeeds with probability > 0.99, then Algorithm 2 succeeds with probability > 0.9. If n is the bit-size of the RSA modulus, each sub-routine runs with a gate count $\mathcal{O}\left(n^3\right)$, depth in $\mathcal{O}\left(n^2 \log^3 n\right)$ and $\frac{n}{2} + o(n)$ qubits. Using $s = \mathcal{O}(\log n)$, the entire algorithm runs with an average gate count $\mathcal{O}\left(n^3 \log n\right)$ and $\frac{n}{2} + \mathcal{O}\left(\frac{n}{\log n}\right)$ qubits. The classical computation is polynomial in n.*

Proof. The complexity of the algorithm is immediately deduced from Theorem 2 and [10]. For the correctness, we prove it in two steps.

Consider first a version of Algorithm 2 in which we have replaced our multi-product circuit by a circuit that computes exactly $\beta \cdot ((G^x A^{-y} \bmod N) \bmod 2^r)$,

where β is the r-bit mask that has been selected uniformly at random. Then by Theorem 3, under Heuristic 1, this version offers the guarantee of measuring the same number of "good" outputs if we eliminate the "0" outputs, which appear in proportion $1/2$.

For a given μ, the probability to measure new zeroes (not those which would have already been output by the uncompressed subroutine) is $1/2$. Thus, we need on average 2μ iterations of the compressed subroutine to obtain μ non-zero measurement outputs. Furthermore, the probability that more than 4μ runs are necessary is small. Indeed, assume that we make 4μ runs, they will contain 2μ non-zero results on average. By a Chernoff bound, the probability that they contain less than μ non-zero results is bounded by $\exp(-(1/2)^2 2\mu/2)$. Starting with $\mu = 20$, this is smaller than 0.007, so the probability of success is greater than 0.993. We use this bound to set the success probability of our circuit in Step 4. The measurement results follow the distribution expected by Ekerå-Håstad, so we can run the classical post-processing unchanged and factor N.

In a second step, we replace the computation of $(G^x A^{-y} \bmod N) \bmod 2^r$ by our circuit, which introduces errors. Under Heuristic 2, we can make the probability of error arbitrarily low. In fact, we can ensure the constraint $8\sqrt{4\mu}\sqrt{p} < 0.08$ where μ is as large as $\mathcal{O}(\log n)$, as this requires $\mathcal{O}(\log \log n)$ additional space only in Heuristic 3.

We can then use Lemma 3. We have established that the "ideal" version of the algorithm succeeds with probability $\geq 0.99 - 0.007$. As there are 4μ measurements at most, the new probability of success is lower bounded by: $0.99 - 0.007 - 8\sqrt{4\mu}\sqrt{p} \geq 0.99 - 0.007 - 0.08 \geq 0.9$. □

Case of General Integers. In the case of general integers, our method can reduce the space to $n + o(n)$ as follows[5].

First, one uses the classical reduction from [11], which factors an integer N completely given the order of a random element G in $\mathbb{Z}_N^*$. To solve this order-finding problem, one uses Seifert's variant of Shor's algorithm [33] combined with the analysis of App. A in [12]. At this point, the input register length is of $m + \lceil m/s \rceil$ where $m \leq n$ is an upper bound on the bit-length of the order of G, and about s runs are required to find the entire order.

5.3 Application to Discrete Logarithms

Although the factorization of RSA moduli is the most prominent application, our method gives a stronger advantage for computing Discrete Logarithms with short exponents in safe-prime groups.

When P is a prime and $Q = (P-1)/2$ is also prime, a safe-prime group is a (cyclic) subgroup of $\mathbb{Z}_P^*$ of order Q, where the computational Diffie-Hellman problem is expected to be hard. Usually $\log_2 P$ is large, but the exponent can be made quite small. Indeed, subexponential classical algorithms for discrete logarithms depend on the entire group size; when the discrete logarithm is of

[5] The details that follow here were explained to us by Martin Ekerå.

size $d \ll \log_2 P$, the best classical algorithms run in $\mathcal{O}\big(2^{d/2}\big)$. As an example, for $\log_2 P = 2048$, $d = 224$ is sufficient to offer 112 bits of security.

Since the exponentiation in $\mathbb{Z}_P^*$ also reduces to computing a modular multi-product, the proof of Theorem 2 carries through.

Corollary 1. *Consider a Discrete Logarithm instance in $\mathbb{Z}_P^*$, of bit-size d. Under Heuristic 1 and Heuristic 2, if the Ekerå-Håstad algorithm running with parameters s and μ succeeds with probability > 0.99, then Algorithm 2 succeeds with probability > 0.9. Each sub-routine runs with a gate count $\widetilde{\mathcal{O}}\big(d^2 \log P\big)$, and $d + o(d) + \mathcal{O}(\log P)$ qubits. The classical computation time is polynomial.*

As a comparison, solving this short discrete logarithm problem would have previously required to compute d multiplications in $\mathbb{Z}_P^*$. The total gate count would be $\mathcal{O}\big(d \log^2 P\big)$ using schoolbook multiplications or $\mathcal{O}\big(d(\log P)^{1.29}\big)$ using the circuits of [24].

6 Applications

In this section, we estimate the resources required by our algorithm for factoring RSA moduli and computing short discrete logarithms. In both cases, we reuse parameters for the Ekerå-Håstad subroutine (input register size and number of measurements) given by Ekerå in [10].

Note that the optimizations made in [10] are stronger than in [15], as Ekerå simulates a discretized histogram of measurement outputs, and does not only use the "good" ones. Formally, our compression theorem (Theorem 3) does not prove that the entire histogram is preserved and rescaled after compression, as we only considered the set of "good" outputs. However, we could generalize this theorem using multiple sets of outputs; if necessary, we could increase r slightly, which has a minor consequence on the number of qubits. As a consequence we believe that this approach is sound.

Since Algorithm 1 is a classical algorithm, we first implemented and tested it with the RSA-2048 instance of the RSA factoring challenge. Our implementation is quite slow, and computes a random output in about one second on a desktop computer. This allowed us to verify the correctness of our algorithm and of Heuristic 3 for failure probabilities around 2^{-6}.

6.1 Quantum Implementation of Modular Multi-Product

We used the Qiskit library [29] to implement the computation of $[A_X]_p$ for a prime p in the RNS (the most costly component of Algorithm 1), following the blueprint of Lemma 9. Our implementation is optimized for RSA-2048 and might certainly be improved by considering other trade-offs between gate count, depth and qubits.

Given a list $a_0 = [A_0]_p, \ldots, a_{m-1} = [A_{m-1}]_p$ of precomputed values, we construct the circuit that on input $(x_0, \ldots, x_{m-1})$ returns the multi-product $\big[\prod a_i^{x_i}\big]_p$. We first precompute the discrete logarithms of the a_i relative to a

generator g of $\mathbb{Z}_p^*$, in order to reduce to a *multi-sum* $\sum_i (x_i \alpha_i)$. After computing the multi-sum, we compute an exponentiation of g by $\sum_i (x_i \alpha_i)$ modulo p.

The multi-sum is implemented with the strategy of Lemma 9, which reduces it to several *multi-bit sums* which compute the Hamming weight of a bit-string. The inputs are divided in groups of 15 bits, and a 4-bit incrementor circuit is used to update each group. The results are summed using a tree of adders.

Since p is typically of 19 to 24 bits, the modular exponentiation that follows can be implemented using windowed arithmetic. We cut the exponent x into three parts: $x = x_1 + 2^7 x_2 + 2^{14} x_3$, and we compute $[g^{x_1}]_p, [(g^{2^7})^{x_2}]_p$ and $[(g^{2^{14}})^{x_3}]_p$ using the table lookup circuit of [1]. Afterwards, we do non-modular multiplications and Euclidean divisions to reduce modulo p.

While the exact costs of the modular multi-product depend on the a_i and on p, we observed that the variations are quite small at these scales; the total counts reported are averaged over 20 to 30 random instances.

Number of Qubits. The number of qubits required by the multi-sum circuit varies depending on the α_i, since the multi-bit sums span only the bits of the d_i that are "1" at specific positions. When $m \geq 1000$ (RSA instances) the probability that an m-bit string has Hamming weight bigger than $1.3m/2$ is lower than $\exp(-0.3^2/(2 + 0.3) \times m/2)$ using a multiplicative Chernoff bound, which is smaller than $\exp(-19.6)$. This ensures that this case never happens for all multi-bit sums used throughout Algorithm 1. Therefore, when estimating the number of qubits (which needs to be an upper bound over all RNS primes), we consider that the multi-bit sums will have $0.65m$ inputs at most. When $m < 500$ (discrete logarithm instances), we simply considered that the multi-bit-sums would have m inputs at most.

Full Quantum Circuit. The quantum circuit follows the layout given in Algorithm 1. We need to keep track of the "accumulator" registers qM ($u + \lceil \log_2 q_M \rceil + 1$ qubits), QN ($u' + r$ qubits) and of the Result register (r qubits). For each prime of the RNS, we will compute a modular multi-product, update the accumulators (with negligible cost), and then uncompute it. When we have computed Result, we uncompute qM and QN. This means that the number of multi-product circuits is 4 times the number of primes in the RNS.

6.2 Parametrizations and Gate Counts

In Section Cwe give the parameters of Algorithm 2 for several RSA and DL instances, and corresponding resource estimates. We detail here the cases of RSA-2048 and a 224-bit DL in a safe-prime group of 2048 bits.

For RSA-2048, following Table 3 in [10], $s = 17$ and $\mu = 20$ measurement results are enough to ensure 0.99 success probability for the post-processing. This yields $m = 1146$. There are 72 199 primes in the RNS, allowing to represent numbers of 1146×2048 bits. A probability of failure $p = 2^{-20}$ for the circuit satisfies the condition $8\sqrt{4\mu}\sqrt{p} < 0.08$, which ensures a total success probability of at least 0.9 for Algorithm 2. To achieve $p = 2^{-20}$, we will use $\nu = 20$ additional

bits for Heuristic 3 (truncation of the sum), the remaining sources of errors being negligible. From the constraints in Section 4, we obtain that the register for q_M will be of size 115 bits, the one for Q_N of size 63 bits, and the result has 22 qubits. Thus the circuit contains at least $115 + 63 + 22 = 200$ ancillas.

Our implementation of multi-product contains in addition 384 ancillas, and takes $2^{17.4}$ Toffoli gates. The full circuit contains 1730 qubits including 584 workspace ancillas, and $2^{35.54}$ Toffoli gates. We need 40 measurements on average, i.e., a total of $2^{40.87}$ Toffolis (almost a thousand times bigger than [19]).

For the 224-bit DL instance, following Table 2 in [10], an input register of 224 bits is enough, and we need 10 measurements. There are 16 526 primes in the RNS, as we need to represent numbers of 224×2048 bits only. The sizes of q_M and Q_N are marginally smaller. Our multi-product circuit requires 206 ancillas, and the full circuit has 684 qubits. The Toffoli count is also significantly smaller, at $2^{31.58}$, for a total $2^{35.9}$ when we multiply it by the 20 average measurement results.

DL if the Group Order is Known. If the order of the subgroup for the short DL instance is known, one can use Shor's algorithm directly, as suggested in [19]. Our compression technique is directly applicable to Shor's algorithm; only two measurement results are required on average instead of one, so we only need to run the circuit twice. The input register size is increased to $2d$, but the total number of gates decreases significantly.

7 Conclusion

In this paper, we reduced the number of logical qubits for quantum factoring and discrete logarithms in $\mathbb{Z}_N^*$, at $d + o(d) + o(\log N)$ when d is the bit-size of the DL. This allows to factor n-bit RSA moduli with $\frac{n}{2} + o(n)$ qubits. In particular, for RSA-2048, we could propose a circuit with 1730 qubits. As a comparison, a state-of-the-art quantum circuit for computing elliptic curve discrete logarithms (ECDLP) given in [21] requires 2124 qubits for a 256-bit instance, though the gate count is much smaller.

This result follows from a purely classical algorithm, based on the RNS, which realizes the *compression* initially proposed by May and Schlieper [25] for the Ekerå-Håstad algorithm [15,10]. The correctness of this algorithm depends on two heuristics. The first one, which is related to the distribution of the outputs of an exponentiation modulo N, could certainly be simplified, and we leave this as an interesting open question. In fact, it was suggested by a reviewer that one could make the value of G vary and average over it, allowing to retrieve a family of functions instead of a single one. This could bring us back closer to the compression result of May and Schlieper.

For the case of factoring, our arithmetic circuit contains $\mathcal{O}(n^3)$ gates. While the constant factor in the $\mathcal{O}$ is small according to our estimates, it remains higher than state-of-the art benchmarks for quantum factoring [19]. The difference for a single run is of a factor 2^5, and multiple runs are required in our

case. However, we believe that our initial gate count estimates could be followed by more advanced optimizations, similarly to the improvements undergone by Shor's algorithm in the last decades.

Finally, another prominent target of Shor's algorithm is the discrete logarithm problem on elliptic curves. Unfortunately, the approach that we used in this paper uses specific tools of integer arithmetic (the RNS, Chinese remaindering) which simply do not exist in elliptic curve groups. Whether there is another way to compress their workspace register (while keeping the circuit size polynomial in n) is an interesting open question.

Acknowledgments. The authors would like to express their gratitude to Martin Ekerå for many insightful comments and remarks, which improved significantly the first version of this paper. We thank Paul Kirchner and anonymous reviewers for helpful comments.

We also thank Daniel J. Bernstein for pointing us to the references [5,6] for the Explicit CRT. A.S. wants to thank Xavier Bonnetain for discussions on Shor's algorithm. This work has been supported by the French Agence Nationale de la Recherche through the France 2030 program under grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

References

1. Babbush, R., Gidney, C., Berry, D.W., Wiebe, N., McClean, J., Paler, A., Fowler, A., Neven, H.: Encoding electronic spectra in quantum circuits with linear t complexity. Physical Review X **8**(4), 041015 (2018). https://doi.org/10.1103/PhysRevX.8.041015
2. Beame, P., Cook, S.A., Hoover, H.J.: Log depth circuits for division and related problems. SIAM J. Comput. **15**(4), 994–1003 (1986). https://doi.org/10.1137/0215070
3. Beauregard, S.: Circuit for Shor's algorithm using $2n + 3$ qubits. arXiv preprint quant-ph/0205095 (2002)
4. Bennett, C.H.: Time/space trade-offs for reversible computation. SIAM J. Comput. **18**(4), 766–776 (1989). https://doi.org/10.1137/0218053
5. Bernstein, D.J.: Multidigit modular multiplication with the explicit chinese remainder theorem. Chapter 4 in "Detecting perfect powers in essentially linear time, and other studies in computational number theory", Ph.D. dissertation, University of California at Berkeley (1995), https://cr.yp.to/papers/mmecrt-19950518-retypeset20220326.pdf
6. Bernstein, D.J., Sorenson, J.P.: Modular exponentiation via the explicit chinese remainder theorem. Math. Comput. **76**(257), 443–454 (2007). https://doi.org/10.1090/S0025-5718-06-01849-7
7. Bernstein, E., Vazirani, U.V.: Quantum complexity theory. SIAM J. Comput. **26**(5), 1411–1473 (1997)
8. Cleve, R., Watrous, J.: Fast parallel circuits for the quantum fourier transform. In: FOCS. pp. 526–536. IEEE Computer Society (2000). https://doi.org/10.1109/SFCS.2000.892140

9. Draper, T.G.: Addition on a quantum computer. arXiv preprint quant-ph/0008033 (2000)
10. Ekerå, M.: On post-processing in the quantum algorithm for computing short discrete logarithms. Des. Codes Cryptogr. **88**(11), 2313–2335 (2020). https://doi.org/10.1007/S10623-020-00783-2
11. Ekerå, M.: On completely factoring any integer efficiently in a single run of an order-finding algorithm. Quantum Inf. Process. **20**(6), 205 (2021). https://doi.org/10.1007/S11128-021-03069-1, https://doi.org/10.1007/s11128-021-03069-1
12. Ekerå, M.: Quantum algorithms for computing general discrete logarithms and orders with tradeoffs. J. Math. Cryptol. **15**(1), 359–407 (2021). https://doi.org/10.1515/JMC-2020-0006, https://doi.org/10.1515/jmc-2020-0006
13. Ekerå, M.: On the success probability of the quantum algorithm for the short DLP. CoRR **abs/2309.01754** (2023). https://doi.org/10.48550/ARXIV.2309.01754, https://doi.org/10.48550/arXiv.2309.01754
14. Ekerå, M., Gärtner, J.: Extending regev's factoring algorithm to compute discrete logarithms pp. 211–242 (2024)
15. Ekerå, M., Håstad, J.: Quantum algorithms for computing short discrete logarithms and factoring RSA integers. In: PQCrypto. Lecture Notes in Computer Science, vol. 10346, pp. 347–363. Springer (2017). https://doi.org/10.1007/978-3-319-59879-6_20
16. Gidney, C.: Factoring with $n+2$ clean qubits and $n-1$ dirty qubits. arXiv preprint arXiv:1706.07884 (2017)
17. Gidney, C.: Approximate encoded permutations and piecewise quantum adders. arXiv preprint arXiv:1905.08488 (2019)
18. Gidney, C.: Windowed quantum arithmetic. arXiv preprint arXiv:1905.07682 (2019)
19. Gidney, C., Ekerå, M.: How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. Quantum **5**, 433 (2021). https://doi.org/10.22331/q-2021-04-15-433
20. Griffiths, R.B., Niu, C.S.: Semiclassical fourier transform for quantum computation. Physical Review Letters **76**(17), 3228 (1996). https://doi.org/10.1103/PhysRevLett.76.3228
21. Häner, T., Jaques, S., Naehrig, M., Roetteler, M., Soeken, M.: Improved quantum circuits for elliptic curve discrete logarithms. In: PQCrypto. Lecture Notes in Computer Science, vol. 12100, pp. 425–444. Springer (2020). https://doi.org/10.1007/978-3-030-44223-1_23
22. Häner, T., Roetteler, M., Svore, K.M.: Factoring using $2n+2$ qubits with toffoli based modular multiplication. arXiv preprint arXiv:1611.07995 (2016)
23. Harvey, D., Van Der Hoeven, J.: Integer multiplication in time $\mathcal{O}(n \log n)$. Annals of Mathematics **193**(2), 563–617 (2021)
24. Kahanamoku-Meyer, G.D., Yao, N.Y.: Fast quantum integer multiplication with zero ancillas (2024)
25. May, A., Schlieper, L.: Quantum period finding is compression robust. IACR Trans. Symmetric Cryptol. **2022**(1), 183–211 (2022). https://doi.org/10.46586/TOSC.V2022.I1.183-211
26. Montgomery, P.L., Silverman, R.D.: An FFT extension to the $p-1$ factoring algorithm. Mathematics of Computation **54**(190), 839–854 (1990)
27. Nielsen, M.A., Chuang, I.: Quantum computation and quantum information (2002)
28. Pollard, J.M.: A monte carlo method for factorization. BIT Numerical Mathematics **15**, 331–334 (1975). https://doi.org/10.1007/bf01933667

29. Qiskit contributors: Qiskit: An open-source framework for quantum computing (2023). https://doi.org/10.5281/zenodo.2573505
30. Ragavan, S., Vaikuntanathan, V.: Space-efficient and noise-robust quantum factoring **14925**, 107–140 (2024). https://doi.org/10.1007/978-3-031-68391-6_4, https://doi.org/10.1007/978-3-031-68391-6_4
31. Regev, O.: An efficient quantum factoring algorithm. CoRR **abs/2308.06572** (2023)
32. Rötteler, M., Steinwandt, R.: A quantum circuit to find discrete logarithms on ordinary binary elliptic curves in depth $o(\log^2 n)$. Quantum Inf. Comput. **14**(9-10), 888–900 (2014). https://doi.org/10.26421/QIC14.9-10-11
33. Seifert, J.: Using fewer qubits in shor's factorization algorithm via simultaneous diophantine approximation. In: CT-RSA. Lecture Notes in Computer Science, vol. 2020, pp. 319–327. Springer (2001). https://doi.org/10.1007/3-540-45353-9_24
34. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172
35. Simon, D.R.: On the power of quantum computation. SIAM J. Comput. **26**(5), 1474–1483 (1997)
36. Takahashi, Y., Kunihiro, N.: A quantum circuit for Shor's factoring algorithm using $2n + 2$ qubits. Quantum Information & Computation **6**(2), 184–192 (2006)
37. Zalka, C.: Fast versions of Shor's quantum factoring algorithm. arXiv preprint quant-ph/9806084 (1998)
38. Zalka, C.: Shor's algorithm with fewer (pure) qubits. arXiv preprint quant-ph/0601097 (2006)

Appendix

A Relation with Circuit Complexity

Computing the "compressed" modular exponentiation $(a^x \bmod N) \bmod 2^r$ in $o(n)$ space can be done by combining several results of circuit complexity in a black-box way, although this does not seem to have been noticed before in the context of Shor's algorithm. However, this may lead to a large polynomial, which will be useless in practice: a dedicated analysis is required to bring this polynomial down to a reasonable $\mathcal{O}(n^3)$.

Since a is a constant, modular exponentiation by x can be first reduced to a "multi-product" of n integers. In [2] Beame, Cook and Hoover showed how to perform a product of n integers in logarithmic depth, as well as modular reduction: this was the basis of Cleve and Watrous' log-depth algorithm for factoring, which also introduced a log-depth implementation of the Quantum Fourier Transform [8]. These techniques use the RNS.

Since the circuit for multi-product has depth $\mathcal{O}(\log n)$, any output bit can be written as the root node in a binary tree of size $n^{\mathcal{O}(\log n)} = \mathsf{poly}(n)$, where the leaves are the input bits (in our case, the control bits for the exponentiation in Shor's algorithm) and each node specifies a Boolean operation. All the nodes in the tree (and *a fortiori* the root) can be computed in time $\mathsf{poly}(n)$ and

using $\mathcal{O}(\log n)$ space using a depth-first exploration. Using Bennett's time-space tradeoff [4], we can make this computation reversible, it will still run in time $\mathsf{poly}(n)$ and using space $\mathcal{O}\big(\log^2 n\big)$. This applies as well to discrete logarithms in $\mathbb{Z}_P^*$, which essentially rely on the same circuit.

In particular, since *any* output bit can be produced in time $\mathsf{poly}(n)$, we can also produce all of them in sequence in time $\mathsf{poly}(n)$. An interesting consequence is that we can use the May-Schlieper compression technique [25] as follows.

Theorem 5. *There exists a quantum algorithm factoring RSA moduli using* $\frac{n}{2} + o(n)$ *qubits and* $\mathsf{poly}(n)$ *gates.*

Proof. We use Theorem 1, combined with [15] (see Section 3.2). We use the following hash function family:

$$\{x \mapsto \beta \cdot x, \beta \in \{0,1\}^n\} \ .$$

Before running each compressed circuit, we select β uniformly at random, and hard-code it into the circuit. This means that we need to compute $\mathcal{O}(n)$ bits of the output (at certain positions depending on β) and XOR them. Obviously this can be done in $\mathcal{O}(\log n)$ depth, so the whole circuit takes $\mathsf{poly}(n)$ gates. □

While we are interested in computing multi-products, which are more relevant to quantum factoring, we note that the RNS was also used for the case of modular exponentiation by Bernstein and Sorenson [6]. These results, which concern depth optimizations on parallel machines, also rely on the *explicit* Chinese Remainder Theorem.

Regarding discrete logarithms on elliptic curves, one can already notice that the best depth available in the literature is $\mathcal{O}\big(\log^2 n\big)$ [32], which leads this generic compression to a superpolynomial complexity $2^{\mathcal{O}\left(\log^2 n\right)} = n^{\mathcal{O}(\log n)}$. The main difference with discrete logarithm in $\mathbb{Z}_p^*$ seems to be the availability of the RNS. For this reason, achieving a polynomial-time workspace compression for the case of elliptic curves remains an open question.

B Proof of Theorem 3

As recalled in Section 3.3, May and Schlieper [25] proved that if a quantum period-finding subroutine has the *cancellation property*, then compressing this subroutine with a universal hash function family only rescales the distribution of measurement outputs y (when measuring the input register). In this section, we prove an alternative theorem that allows to rely only on Heuristic 1, i.e., on the randomness of the output of the modular multi-product circuit. Here r will be a well-chosen constant.

Fixed Function Analysis. In the case of Ekerå-Håstad, let us first consider the uncompressed subroutine, where the function f is defined by $f(x_1, x_2) =$

$G^{x_1}A^{-x_2} \bmod N$. Like in [25], we define $w_{y,f(x)}$ as the amplitudes in the output of the subroutine Q_f^{period}:

$$\sum_{y\in\{0,1\}^m} \sum_{f(x)\in Im(f)} w_{y,f(x)} \left|y\right\rangle \left|f(x)\right\rangle \quad . \tag{30}$$

By a change of variable $z = f(x)$ we can rewrite this state as:

$$\sum_{y\in\{0,1\}^m} \sum_{z\in\{0,1\}^n} w_{y,z} \left|y\right\rangle \left|z\right\rangle \quad . \tag{31}$$

The *cancellation property* is satisfied:

$$\forall y \neq 0, \sum_{z\in\{0,1\}^n} w_{y,z} = 0 \quad . \tag{32}$$

The probability to measure a given y in the non-compressed state (Equation 31) is:

$$p_N(y) := \sum_{z\in\{0,1\}^n} |w_{y,z}|^2 \quad . \tag{33}$$

Here the subscript N in $p_N(y)$ indicates that this quantity depends only on N, since the definition of the function f depends only on N (we assume that there is a canonical choice of G). So far, we have considered the actual uncompressed algorithm, whose correctness follows from the analysis of [15,10]. The amplitudes $w_{y,z}$ (where we omit the subscript N) are those of this algorithm.

Randomizing the Function (Uncompressed). We introduce now a modification in the function. We compose f with a random permutation of $\mathbb{Z}_N^*$, which is extended to $\{0,1\}^n$, and denoted as h. This modifies the output state into:

$$\sum_{y\in\{0,1\}^m} \sum_{f(x)\in Im(f)} w_{y,f(x)} \left|y\right\rangle \left|h(f(x))\right\rangle = \sum_{y\in\{0,1\}^m} \sum_{z\in\{0,1\}^n} w_{y,z} \left|y\right\rangle \left|h(z)\right\rangle \quad . \tag{34}$$

We define as $p_{N,h}(y)$ the probability to measure y in this state. Then we can notice that for any h, $p_{N,h}(y) = p_N(y)$.

Compressing the Function. We introduce now a compression of the function. We take the dot-product of the output with a mask $\beta' = \beta|0^{n-r}$ where β is a non-zero r-bit mask. The compressed subroutine will now produce the output:

$$\sum_{y\in\{0,1\}^m} \left(\sum_{\substack{z\in\{0,1\}^n \\ (\beta|0^{n-r})\cdot h(z)=0}} w_{y,z} \left|y\right\rangle \left|0\right\rangle + \sum_{\substack{z\in\{0,1\}^n \\ (\beta|0^{n-r})\cdot h(z)=1}} w_{y,z} \left|y\right\rangle \left|1\right\rangle \right) \quad . \tag{35}$$

Again, notice that $w_{y,z}$ have a fixed definition which is from the original Ekerå-Håstad algorithm, while h is a varying random permutation. The probability to obtain a given y when measuring the state in Equation 35 is:

$$p'_{N,h}(y) := \frac{1}{2^r - 1} \sum_{\beta \neq 0} \left(\left| \sum_{(\beta|0^{n-r}) \cdot h(z) = 0} w_{y,z} \right|^2 + \left| \sum_{(\beta|0^{n-r}) \cdot h(z) = 1} w_{y,z} \right|^2 \right) \quad . \tag{36}$$

While $p'_{N,h}(y)$ depends on N and h, we also introduce $p'_N(y)$, which is the probability to measure y when running the compressed subroutine with f only (i.e., in our algorithm).

In the following, we prove that $p'_{N,h}(y)$ is very close to $\frac{1}{2}p_N(y)$ when h is chosen uniformly at random. Intuitively, the role of h is to partition randomly the terms $w_{y,z}$ into the two subsets $(\beta|0^{n-r}) \cdot h(z) = 0$ or $(\beta|0^{n-r}) \cdot h(z) = 1$.

Lemma 10. *Let N be fixed, let $y \neq 0$ be fixed, and h be a uniformly random permutation of $\mathbb{Z}_N^*$. For any $k > 0$:*

$$\Pr_h \left(|\frac{1}{2}p_N(y) - p'_{N,h}(y)| \geq \frac{k p_N(y)}{\sqrt{(2^r - 1)}} \right) \leq \frac{1}{k^2} \quad . \tag{37}$$

Proof. We first notice that, by the cancellation property:

$$\left| \sum_z w_{y,z} \right|^2 = 0 = \sum_z |w_{y,z}|^2 + \sum_{z \neq z' \in \{0,1\}^n} w_{y,z} \overline{w_{y,z'}} \quad . \tag{38}$$

Let $h_\beta := (\beta|0^{n-r}) \cdot h$ for any $\beta \neq 0$. By developing $p'_{N,h}(y)$ we have:

$$\begin{aligned}
p'_{N,h}(y) &= \frac{1}{2^r - 1} \sum_{\beta \neq 0} \left(\sum_{z \in \{0,1\}^n} |w_{y,z}|^2 + \sum_{\substack{z \neq z' \\ z,z' \in h_\beta^{-1}(0)}} w_{y,z} \overline{w_{y,z'}} + \sum_{\substack{z \neq z' \\ z,z' \in h_\beta^{-1}(1)}} w_{y,z} \overline{w_{y,z'}} \right) \\
&= \frac{1}{2^r - 1} \sum_{\beta \neq 0} \left(\frac{1}{2} p(y) + \frac{1}{2} \sum_{z \in \{0,1\}^n} |w_{y,z}|^2 \right. \\
&\quad \left. + \sum_{\substack{z \neq z' \\ z,z' \in h_\beta^{-1}(0)}} w_{y,z} \overline{w_{y,z'}} + \sum_{\substack{z \neq z' \\ z,z' \in h_\beta^{-1}(1)}} w_{y,z} \overline{w_{y,z'}} \right) \\
&= \frac{1}{2} p_N(y) + \frac{1}{2(2^r - 1)} \sum_{\beta \neq 0} \left(\sum_{\substack{z \neq z' \\ z,z' \in h_\beta^{-1}(0)}} w_{y,z} \overline{w_{y,z'}} + \sum_{\substack{z \neq z' \\ z,z' \in h_\beta^{-1}(1)}} w_{y,z} \overline{w_{y,z'}} \right. \\
&\quad \left. - \sum_{\substack{z \in h_\beta^{-1}(0) \\ z' \in h_\beta^{-1}(1)}} w_{y,z} \overline{w_{y,z'}} - \sum_{\substack{z \in h_\beta^{-1}(1) \\ z' \in h_\beta^{-1}(0)}} w_{y,z} \overline{w_{y,z'}} \right) \quad .
\end{aligned}$$

We will now introduce the following random variables (y and N being fixed, the randomness here is the choice of h): $X_{\beta,z} = h_\beta(z)$. Observe that for any pair z, z' with $z \neq z'$: $(-1)^{X_{\beta,z}+X_{\beta,z'}}$ takes the value 1 if z and z' have the same image by h_β and -1 otherwise. This allows us to rewrite nicely the expression of $p'_{N,h}$:

$$p'_{N,h}(y) = \frac{1}{2}p_N(y) + \frac{1}{2(2^r-1)} \sum_{\beta \neq 0} \left(\sum_{\substack{z,z' \in \{0,1\}^n \\ z \neq z'}} (-1)^{X_{\beta,z}+X_{\beta,z'}} w_{y,z}\overline{w_{y,z'}} \right) . \quad (39)$$

Define $V_y := p'_{N,h}(y) - \frac{1}{2}p_N(y)$. In the proof of Theorem 7 in [25], this additional term would be 0. This is not the case here; however we will still manage to bound it. First of all, let us rewrite V_y as a sum of real terms, by putting together the pairs z, z' and z', z: we will now have a sum over *unordered pairs* of $\{0,1\}^n$:

$$V_y = \frac{1}{(2^r-1)} \sum_{\beta \neq 0} \left(\sum_{\substack{z,z' \in \{0,1\}^n \\ \text{unordered pairs}}} (-1)^{X_{\beta,z}+X_{\beta,z'}} Re(w_{y,z}\overline{w_{y,z'}}) \right) . \quad (40)$$

Let $Y_{\beta,z,z'} = (-1)^{X_{\beta,z}+X_{\beta,z'}}$. We have the following properties:

1. For any (β, z, z'), $\mathbb{E}_h[Y_{\beta,z,z'}] = 0$
2. For any $(\beta_1, z_1, z'_1) \neq (\beta_2, z_2, z'_2)$, Y_{β_1,z_1,z'_1} and Y_{β_2,z_2,z'_2} are independent

The first one is easy to prove. The second is more technical, as we have to consider different cases. First, if $\beta_1 \neq \beta_2$, then h_{β_1} and h_{β_2} are independent random functions, so the random variables Y_{β_1,z_1,z'_1} and Y_{β_2,z_2,z'_2} become independent as well. Second, even if $\beta_1 = \beta_2$, if the pairs are disjoint, then the variables $X_{\beta_1,z_1}, X_{\beta_1,z'_1}, X_{\beta_2,z_2}, X_{\beta_2,z'_2}$ are all pairwise independent, which gives the property. The remaining case is when $\beta_1 = \beta_2$ and the pairs share an element: w.l.o.g., we can write the variables as: $(-1)^{X_{\beta,z_1}+X_{\beta,z_2}}$ and $(-1)^{X_{\beta,z_2}+X_{\beta,z_3}}$. Then, the probability to obtain any pair $\{-1, 1\}$ is still $\frac{1}{4}$.

In particular, even in the latter case, $\mathbb{E}_h[Y_{\beta,z_1,z_2}Y_{\beta,z_2,z_3}] = \mathbb{E}_h[Y_{\beta,z_1,z_3}] = 0$.

Thanks to this pairwise independence of the variables in the sum, we can now compute the variance of V_y:

$$\begin{aligned}
\mathrm{Var}_h(V_y) &= \sum_{\beta,z,z'} \mathrm{Var}_h\left(\frac{1}{(2^r-1)} Y_{\beta,z,z'} Re(w_{y,z}\overline{w_{y,z'}}) \right) \\
&= \frac{1}{(2^r-1)^2} \sum_{\beta,z,z'} \left(Re(w_{y,z}\overline{w_{y,z'}})\right)^2 \\
&\leq \frac{1}{(2^r-1)} \sum_{\substack{z,z' \in \{0,1\}^n \\ \text{unordered pairs}}} |w_{y,z}\overline{w_{y,z'}}|^2 \\
&\leq \frac{1}{(2^r-1)} \sum_{z,z' \in \{0,1\}^n} |w_{y,z}\overline{w_{y,z'}}|^2 \leq \frac{p(y)^2}{(2^r-1)} .
\end{aligned}$$

We can apply Chebyshev's inequality:

$$\forall k > 0, \Pr_h \left(|V_y| \geq \frac{p_N(y)k}{\sqrt{(2^r - 1)}} \right) \leq \frac{1}{k^2} \ . \tag{41}$$

This finishes the proof of the lemma. □

This inequality indicates that, at fixed N, when running the algorithm with $h \circ f$ instead of f, we can expect the compression to work: the probability to measure a fixed y will be close to half of the probability to measure it in the uncompressed algorithm. Our next result extends this to any set $\mathcal{Y}$ of non-zero outputs.

Lemma 11. *Fix N. Consider any non-empty set $\mathcal{Y} \subseteq \{0,1\}^m \backslash \{(0,0,\ldots,0)\}$, and h be a uniformly random permutation of $\mathbb{Z}_N^*$. For any $k > 0$:*

$$\Pr_h \left(\left| \frac{1}{2} p_N(\mathcal{Y}) - p'_{N,h}(\mathcal{Y}) \right| \geq \frac{k p_N(\mathcal{Y})}{\sqrt{(2^r - 1)}} \right) \leq \frac{1}{k^2} \ . \tag{42}$$

Proof. Since the probability to measure $\mathcal{Y}$ can be obtained by: $p_N(\mathcal{Y}) = \sum_{y \in \mathcal{Y}} p_N(y)$ and $p'_{N,h}(\mathcal{Y}) = \sum_{y \in \mathcal{Y}} p'_{N,h}(y)$, the proof of Lemma 10 can be modified by replacing the terms $w_{y,z}\overline{w_{y,z'}}$ by their sums over $\mathcal{Y}$. In particular, we define:

$$V_{\mathcal{Y}} := \sum_{y \in \mathcal{Y}} \left(p'_{N,h}(y) - \frac{1}{2} p_N(y) \right) \ . \tag{43}$$

And we have:

$$V_{\mathcal{Y}} = \frac{1}{2^r - 1} \sum_{\beta \neq 0} \sum_{\substack{z,z' \in \{0,1\}^n \\ \text{unordered pairs}}} Y_{\beta,z,z'} \left(\sum_{y \in \mathcal{Y}} Re(w_{y,z}\overline{w_{y,z'}}) \right)$$

$$\implies \mathrm{Var}_h(V_{\mathcal{Y}}) = \frac{1}{2^r - 1} \sum_{\substack{z,z' \in \{0,1\}^n \\ \text{unordered pairs}}} \left(\sum_{y \in \mathcal{Y}} Re(w_{y,z}\overline{w_{y,z'}}) \right)^2$$

$$\leq \frac{1}{2^r - 1} \sum_{\substack{z,z' \in \{0,1\}^n \\ \text{unordered pairs}}} \left(\sum_{y \in \mathcal{Y}} |w_{y,z}||w_{y,z'}|) \right)^2 \ .$$

In order to bound this, we use the Cauchy-Schwarz inequality, leading to:

$$\mathrm{Var}_h(V_{\mathcal{Y}}) \leq \frac{1}{2^r - 1} \sum_{\substack{z,z' \in \{0,1\}^n \\ \text{unordered pairs}}} \left(\sum_{y \in \mathcal{Y}} |w_{y,z}|^2 \right) \left(\sum_{y \in \mathcal{Y}} |w_{y,z'}|^2 \right)$$

$$\leq \frac{1}{2^r - 1} \left(\sum_z \sum_{y \in \mathcal{Y}} |w_{y,z}|^2 \right)^2 = \frac{p_N(\mathcal{Y})^2}{2^r - 1} \ .$$

This finishes the proof of the lemma. □

As a corollary, by taking $\mathcal{Y}$ to contain all y except 0, we can see that the probability to measure 0 will be close to $\frac{1}{2}$.

Proving the Theorem. Following the analysis of the uncompressed Ekerå-Håstad subroutine [15], one can define a set of "good" outputs $\mathcal{Y}$ and the probability to measure a "good" y is greater than 2^{-3}. The definition of this set depends only on the discrete logarithm, not on N and f.

Our goal is essentially to show that the compressed version also measures good outputs, and that we need only twice as many measurements on average.

Lemma 12 (From [15]). *Fix N. Let $\mathcal{Y}$ be the set of "good" outputs and $p_N(y)$ be defined as above. We have:*

$$p_N(\mathcal{Y}) := \sum_{y \in \mathcal{Y}} p_N(y) \geq 2^{-3} \ .$$

Furthermore, we measure 0 with negligible probability.

Theorem 6. *Let $r = 22$. Under Heuristic 1, with total probability 0.99 over the choice of N, when running our compressed Ekerå-Håstad subroutine:*

- *we measure 0 with probability at most $\frac{1}{2} + 0.01$*
- *we measure a "good" y with probability at least $\frac{1}{16} - 0.01$*

Proof. Let $\mathcal{Y}$ be the set of "good" outputs (which is fixed if the DL is fixed).

Fix N. We do a union bound on the result of Lemma 11, considering both $\mathcal{Y}$ and the set of all nonzero outputs. When h is chosen at random, with probability at least $1 - 2/k^2$ over h, the following holds:

$$\begin{cases} \left| \frac{1}{2} p_N(\mathcal{Y}) - p'_{N,h}(\mathcal{Y}) \right| \leq \frac{k}{\sqrt{2^r - 1}} \implies p'_{N,h}(\mathcal{Y}) \geq \frac{1}{2} p_N(\mathcal{Y}) - \frac{k}{\sqrt{2^r - 1}} \\ \left| p'_{N,h}(0) - \frac{1}{2}(1 + p_N(0)) \right| \leq \frac{k}{\sqrt{2^r - 1}} \implies p_{N,h}(0) \leq \frac{1}{2}(1 + p_N(0)) + \frac{k}{\sqrt{2^r - 1}} \end{cases} \tag{44}$$

We choose $k = 2^4$ so that $1 - 2/k^2 \geq 0.99$. By choosing $r = 22$ we obtain that $2^4/\sqrt{2^r - 1} \leq 0.01$. This proves that:

$$\Pr_{N,h} \left(\begin{cases} p'_{N,h}(\mathcal{Y}) \geq \frac{1}{16} - 0.01 \\ p'_{N,h}(0) \leq \frac{1}{2} + 0.01 \end{cases} \right) \geq 0.99 \ . \tag{45}$$

So far we have studied the behavior of the compressed algorithm running with $h \circ f$, using the behavior of the uncompressed algorithm running with f (which was supposed to work). The last step is to relate this to the compressed algorithm running with f. Here, we will use Heuristic 1.

Under Heuristic 1, when N is chosen at random, the function f is a random periodic function. So the two situations are statistically equivalent:

- Selecting N at random, h at random and running the compressed subroutine with f;
- Selecting N at random, h at random and running the compressed subroutine with $h \circ f$.

In particular, the distributions of $(p'_{N,h}(\mathcal{Y}), p'_{N,h}(0))$ and $(p'_N(\mathcal{Y}), p'_N(0))$ when both h and N are random are identical, and:

$$\Pr_N\left(\begin{cases} p'_N(\mathcal{Y}) \geq \frac{1}{16} - 0.01 \\ p'_N(0) \leq \frac{1}{2} + 0.01 \end{cases}\right) = \Pr_{N,h}\left(\begin{cases} p'_{N,h}(\mathcal{Y}) \geq \frac{1}{16} - 0.01 \\ p'_{N,h}(0) \leq \frac{1}{2} + 0.01 \end{cases}\right) \geq 0.99 \ .$$

□

The enhanced analysis in [10], which we reuse for our benchmarks, considers a more complicated situation in which the possible measurement results y are not only "good" or "not good". However, it is possible to classify them into a number of groups $\mathcal{Y}_i$, and estimate the success of the algorithm based on the probability to fall into one of these groups. In particular, a generalized version of Theorem 3 can show that if the number of groups is small enough, the probability to fall into each of them is close to half of the previous one.

While this does not prove rigorously that the distribution follows exactly the one simulated in [10], discretizing the histogram of the distribution is already what Ekerå does for simulating the output. Therefore, we believe that this simplification is sound.

C Parameters and Gate Counts

C.1 Application to RSA Factorization

The parameters of our algorithm and the corresponding costs are given in Table 2 and Table 3. Recall that:

- $s = \mathcal{O}(\log n)$ is the parameter for Ekerå-Håstad
- $r = 22$ is the size of the result that the circuit computes
- m is the bit-size of the input register for the multi-product
- ν is the number of bits in the truncation of the sum (Heuristic 3), so that the probability of success is $\geq 1 - 2^{-\nu}$
- u is the precision of the fixed-point approximation of $\frac{1}{p}$ in the computation of q_M
- u' is the expected number of bits for carry propagation in the computation of Q_N
- d is the bit-size of the discrete logarithm (for DL instances)

C.2 Application to Discrete Logarithms

The parameters and counts for Discrete Logarithms are given in Table 4 and Table 5.

Table 2. Parameters for Algorithm 1, for different RSA moduli, for a probability of success > 0.9. The number of measurements for the Ekerå-Håstad algorithm is taken from Table 3 in [10], and then multiplied by 2 ("us") for the average number of measurements in Algorithm 2.

RSA bit-size	2048	3072	4096	6144	8192
$s, \ell = \lceil \frac{n}{2s} \rceil$	17, 61	21, 74	24, 86	31, 100	34, 121
$m = \frac{n}{2} + 2\ell$	1146	1684	2220	3272	4338
Measurements: Ekerå, us (average)	20, 40	24, 48	27, 54	34, 68	37, 74
ν	20	20	21	21	21
Number of primes in the RNS	72 199	150 475	253 632	527 641	892 693
Maximal bit-size of primes in the RNS	21	22	22	23	24
u	57	60	62	65	67
u'	41	42	44	45	46
$\lceil \log_2 q_M \rceil$	57	60	61	64	67
Size of q_M register $= \lceil \log_2 q_M \rceil + u + 1$	115	121	124	130	135
Size of Q_N register $= u' + r$	63	64	66	67	68
Size of result register $= r$	22	22	22	22	22

Table 3. Estimated qubit count, gate counts and depth for a single modular multi-product circuit, a full circuit for Algorithm 1, and our full Algorithm 2, for RSA moduli. The latter takes into account the number of runs that must be performed in the compressed Ekerå-Håstad algorithm, *on average*, given in Table 2.

n		Qubits (incl. ancilla)	Toffoli	CNOT	X	Depth
2048	Single multi-product	1530 (384)	$2^{17.4}$	$2^{16.97}$	$2^{11.07}$	$2^{15.22}$
	Full circuit	1730 (584)	$2^{35.54}$	$2^{35.11}$	$2^{29.21}$	$2^{33.36}$
	Full algorithm	1730 (584)	$2^{40.87}$	$2^{40.43}$	$2^{34.53}$	
3072	Single multi-product	2208 (524)	$2^{17.91}$	$2^{17.43}$	$2^{11.07}$	$2^{15.23}$
	Full circuit	2415 (731)	$2^{37.11}$	$2^{36.63}$	$2^{30.27}$	$2^{34.43}$
	Full algorithm	2415 (731)	$2^{42.7}$	$2^{42.21}$	$2^{35.85}$	
4096	Single multi-product	2886 (666)	$2^{18.28}$	$2^{17.77}$	$2^{11.07}$	$2^{15.31}$
	Full circuit	3096 (876)	$2^{38.23}$	$2^{37.72}$	$2^{31.02}$	$2^{35.26}$
	Full algorithm	3096 (876)	$2^{43.99}$	$2^{43.48}$	$2^{36.78}$	
6144	Single multi-product	4211 (939)	$2^{18.8}$	$2^{18.27}$	$2^{11.07}$	$2^{15.31}$
	Full circuit	4430 (1158)	$2^{39.81}$	$2^{39.28}$	$2^{32.08}$	$2^{36.32}$
	Full algorithm	4430 (1158)	$2^{45.9}$	$2^{45.36}$	$2^{38.17}$	
8192	Single multi-product	5556 (1218)	$2^{19.21}$	$2^{18.66}$	$2^{11.06}$	$2^{15.39}$
	Full circuit	5781 (1443)	$2^{40.97}$	$2^{40.42}$	$2^{32.83}$	$2^{37.16}$
	Full algorithm	5781 (1443)	$2^{47.18}$	$2^{46.63}$	$2^{39.04}$	

Table 4. Parameters for Algorithm 1, for discrete logarithms, for a probability of success > 0.9. The number of measurements for the Ekerå-Håstad algorithm is taken from Table 2 in [10], and then multiplied by 2 ("us") for the average number of measurements in Algorithm 2.

$\lceil \log_2 p \rceil, d$	2048, 224	3072, 256	4096, 304	6144, 352	8192, 400
$s, \ell = \lceil \frac{d}{s} \rceil$	7, 32	8, 32	9, 34	10, 36	11, 37
$m = d + 2\ell$	288	320	372	424	474
Measurements: Ekerå, us (average)	10, 20	11, 22	12, 24	13, 26	14, 28
ν	19	19	19	19	20
Number of primes in the RNS	20 744	33 792	51 050	84 690	122 828
Maximal bit-size of primes in the RNS	20	20	20	21	21
u	53	54	56	57	59
u'	38	39	39	40	42
$\lceil \log_2 q_M \rceil$	53	54	55	58	58
Size of q_M register $= \lceil \log_2 q_M \rceil + u + 1$	107	109	112	116	118
Size of Q_N register $= u' + r$	60	61	61	62	64
Size of result register $= r$	22	22	22	22	22

Table 5. Estimated qubit count, gate counts and depth for a single modular multi-product circuit, a full circuit for Algorithm 1, and our full Algorithm 2, for Discrete Logarithm instances. The latter takes into account the number of runs that must be performed in the compressed Ekerå-Håstad algorithm, *on average*, given in Table 2.

n (d)		Qubits (incl. ancilla)	Toffoli	CNOT	X	Depth
2048 (224)	Single multi-product	495 (207)	$2^{15.81}$	$2^{15.65}$	$2^{11.07}$	$2^{15.05}$
	Full circuit	684 (396)	$2^{32.15}$	$2^{31.99}$	$2^{27.41}$	$2^{31.4}$
	Full algorithm	684 (396)	$2^{36.47}$	$2^{36.31}$	$2^{31.73}$	
3072 (256)	Single multi-product	527 (207)	$2^{15.92}$	$2^{15.73}$	$2^{11.07}$	$2^{15.06}$
	Full circuit	719 (399)	$2^{32.96}$	$2^{32.78}$	$2^{28.11}$	$2^{32.11}$
	Full algorithm	719 (399)	$2^{37.42}$	$2^{37.24}$	$2^{32.57}$	
4096 (304)	Single multi-product	579 (207)	$2^{16.07}$	$2^{15.85}$	$2^{11.07}$	$2^{15.06}$
	Full circuit	774 (402)	$2^{33.71}$	$2^{33.49}$	$2^{28.71}$	$2^{32.7}$
	Full algorithm	774 (402)	$2^{38.3}$	$2^{38.07}$	$2^{33.29}$	
6144 (352)	Single multi-product	631 (207)	$2^{16.21}$	$2^{15.96}$	$2^{11.08}$	$2^{15.06}$
	Full circuit	831 (407)	$2^{34.58}$	$2^{34.33}$	$2^{29.45}$	$2^{33.43}$
	Full algorithm	831 (407)	$2^{39.28}$	$2^{39.03}$	$2^{34.15}$	
8192 (400)	Single multi-product	681 (207)	$2^{16.34}$	$2^{16.06}$	$2^{11.07}$	$2^{15.09}$
	Full circuit	885 (411)	$2^{35.25}$	$2^{34.97}$	$2^{29.97}$	$2^{34.0}$
	Full algorithm	885 (411)	$2^{40.05}$	$2^{39.77}$	$2^{34.78}$	