Quantum Factorization with Less Qubits

Jiun-Cheng Jiang,[1, *] YuChao Hsu,[2, 3, †] Jun Liang Tan,[4, ‡] Ran-Yu,Chang,[5, §] Chun-Hua Lin,[1] Peng Kuo Chung,[1] Yi-Kai Lee,[6] and Yun-Hsuan Chen[7]

[1] *Department of Physics and Center for Theoretical Physics, National Taiwan University, Taipei 106319, Taiwan*
[2] *National Center for High-Performance Computing, NARlabs, Hsinchu, Taiwan*
[3] *Cross College Elite Program, National Cheng Kung University, Tainan, Taiwan*
[4] *The University of Queensland, Australia*
[5] *Arete Honor Program with quantum computing major, National Yang Ming Chiao Tung University, Hsinchu, Taiwan*
[6] *Department of Statistics, National Taipei University, New Taipei City, Taiwan*
[7] *Concordia International School Shanghai, Shanghai, China*

Shor's algorithm, introduced in 1994, demonstrated polynomial-time integer factorization, presenting a potential threat to classical cryptography. Despite advances in optimizing the algorithm, its practical application remains hindered by the significant resources required for fault-tolerant quantum computation. In this work, we propose a space-efficient quantum factoring algorithm that reduces qubit requirements while maintaining computational efficiency. By introducing a compressed Eker-Håstad subroutine, we minimize the qubits needed for modular exponentiation and improve feasibility on near-term quantum hardware. The algorithm consists of three stages: **Classical Preprocessing** for input preparation, **Quantum Computation** for period finding using controlled modular multiplication, and **Classical Post-Processing** to recover prime factors. Our approach optimizes qubits, reduces circuit depth, and enhances gate efficiency, making quantum factorization more practical for current quantum architectures.

The implementation is available at: `https://github.com/HSUYUCHAO/2025_MIT-QuantumRing_Remote`

I. INTRODUCTION

In 1994, Shor proposed a groundbreaking quantum algorithm capable of factoring integers and computing discrete logarithms (DL) in polynomial time.[1]Since then numerous researchers have proposed variants of Shor's algorithm and refined its resource cost estimates, aiming to optimize its implementation by reducing the number of required qubits, minimizing gate counts, and lowering circuit depth.[2, 3] Given that Shor's algorithm represents one of the most significant applications of quantum computing in cryptanalysis[4], these studies also seek to determine when quantum computing architectures will reach a level where they pose a practical threat to cryptographic security. However, executing Shor's algorithm on a fault-tolerant quantum computer requires substantial computational resources[5]. Including a large number of logical qubits, deep quantum circuits, and extensive error correction mechanisms. The overhead associated with fault tolerance significantly increases the complexity of implementing Shor's algorithm in practice, making it challenging to realize on near-term quantum hardware[6]. In our work, we present a novel space-efficient quantum factoring algorithm designed to minimize qubit requirements while maintaining computational efficiency. Based on the principles of Shor's algorithm, we introduce a compressed Eker-Hstad subroutine[7], which reduces the number of qubits required for modular exponentiation and improves the feasibility of quantum factorization in near-term quantum hardware. Our approach extends previous advances in space-efficient quantum factoring by further refining qubit optimization techniques, reducing circuit depth, and improving gate efficiency, making quantum cryptanalysis more practical within existing quantum computational constraints.

II. METHOD

In this Hackathon, we replicate the Algorithm proposed in the paper[8], by combining several techniques: 1. The kerå-Håstad variant of Shor's algorithm; 2. A compression technique inspired by May and Schlieper 3. A novel approach for computing truncated modular exponentiation using a Residue Number System (RNS)

The Algorithm designed quantum circuit that implements a "compressed" version of the Eker a-H astad algorithm, which has several key functions:

- Input register of size $n/2 + o(n)$ qubits
- Workspace register of $o(n)$ qubits
- Computation of $h(G^x * A^{-y} \bmod N)$ instead of the full modular exponentiation

The classical reversible circuit computes the least significant bits of $G^x * A^{-y} \bmod N$, where $r = 22$ is a small

[*] r13222039@g.ntu.edu.tw
[†] an4114752@gs.ncku.edu.tw
[‡] junliang.tan@student.uq.edu.au
[§] leo07010@gmail.com

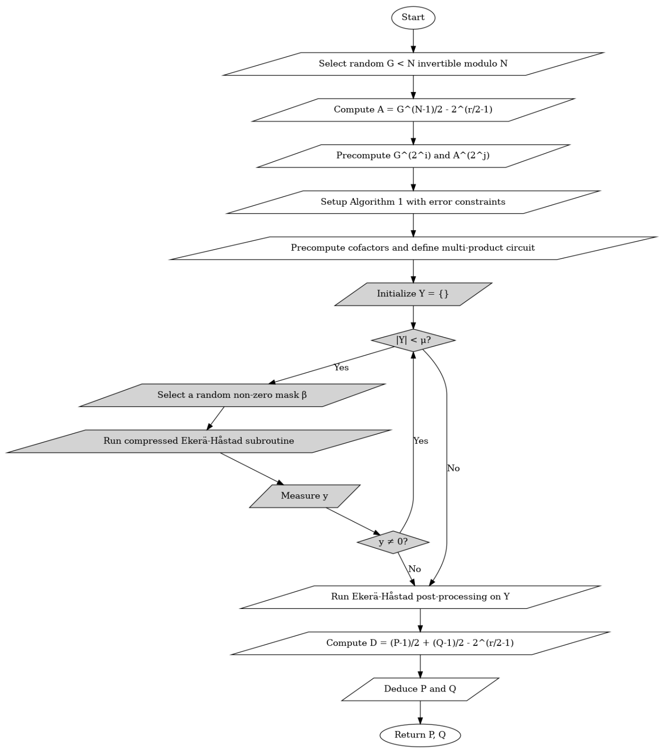
Start
Select random G < N invertible modulo N
Compute A = G^(N-1)/2 - 2^(r/2-1)
Precompute G^(2^i) and A^(2^j)
Setup Algorithm 1 with error constraints
Precompute cofactors and define multi-product circuit
Initialize Y = {}
|Y| < μ?
Yes
Select a random non-zero mask β
Run compressed Ekerå-Håstad subroutine
Measure y
Yes
No
y ≠ 0?
No
Run Ekerå-Håstad post-processing on Y
Compute D = (P-1)/2 + (Q-1)/2 - 2^(r/2-1)
Deduce P and Q
Return P, Q

FIG. 1: Algorithmic flowchart illustrating the compressed Ekerå-Håstad variant of Shor's algorithm. The white sections represent classical preprocessing and post-processing steps, while the gray section denotes the quantum computation phase responsible for extracting periodicity.

constant. The circuit uses a representation of the residual number system, expresses the desired output bits as a large sum of integers of $O(log\ n)$ bits, and computes these integers to reduce the small primes of the multi-product module in the RNS. This introduces a small probability of error from truncation sum, which we assume the error can be made arbitrarily small by heuristic 2, that require multi-run to success with high probability.

In Quantum Subroutines in Fig.2, we initialize total A list number of qubits in control register, $\lceil \log_2(N) \rceil$ qubits as output register and 1 ancillas qubits, and apply hadamard gates to control register, the "compressed Ekerå-Håstad" circuit to compute $h(G^x \cdot A^{-y} \mod N)$, apply new random β mask to the output, inverse Quantum Fourier Transform (QFT) to the control register, then measure the control register, collect measurement$Y$. Each circuit run we uses a new random mask β and collect measurement result in set Y.

In classical Post-processing, for each measurement y in Y, compute the gcd(y,N), where if $1 < gcd(y, N) < N$, add it to the list of factors, and return the list.

A. Algorithm Flowchart Explanation

Figure 1 illustrates the overall structure of our algorithm, which is based on the compressed Ekerå-Håstad variant of Shor's algorithm. The flowchart outlines the sequence of steps required to achieve the determination of modular periods and factorization.

The algorithm can be divided into three main stages following the methodology proposed by Clémence Chevignard:

1. **Classical Preprocessing (White Sections):** The initial stage consists of classical preprocessing steps, where the input parameters are prepared for quantum computation. This includes selecting a random base $G < N$, computing $A = G^{(N-1)/2} - 2^{(r/2-1)}$, and pre-computing the necessary modular exponentiation values such as $G^{(2^i)}$ and $A^{(2^j)}$.

2. **Quantum Computation (Gray Section):** The quantum circuit implementation is responsible for executing the compressed Ekerå-Håstad subroutine. This involves computing $h(G^x * A^{-y} \bmod N)$ instead of performing full modular exponentiation. The quantum register is initialized, and a controlled modular multiplication operation is applied to extract periodicity information. The measurement step determines whether additional iterations are required. As shown in Figure 2, we demonstrate how a quantum circuit is utilized to implement the period-finding method for the semiprime 143, with the final result obtained through measurement.

3. **Classical Post-Processing (White Sections):** The final stage consists of classical post-processing, where the quantum measurement results are analyzed and transformed into a useful factorization output. The post-processing step applies the Ekerå-Håstad heuristic to recover factors P and Q, followed by computing $D = (P-1)/2+(Q-1)/2-2^{(r/2-1)}$. Finally, the algorithm deduces the prime factors of N.

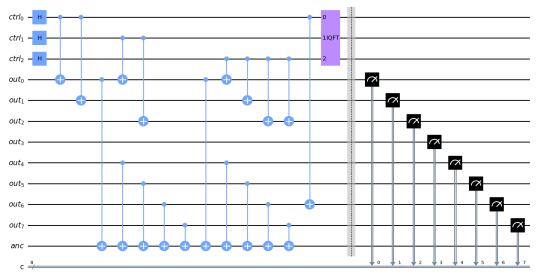

FIG. 2: The circuit implementation of the Controlled Modular Multi-Product approach for period finding of 143.

III. RESULTS

In this section, we present the results of our optimized quantum factorization approach on the Quantum Rings platform.

TABLE I: Experimental results of quantum factorization. Here, N represents semi-primes, P and Q are the factors of N, and the time measurement includes only the quantum computation.

N (bits)	P	Q	# Qubits	# Gates	Time(s)
744647 (20)	907	821	42	677	2844
167659 (18)	431	389	38	544	62
47053 (16)	223	211	34	459	87
11009 (14)	109	101	30	301	17
3127 (12)	59	53	26	270	3
899 (10)	31	29	22	179	6
143 (8)	13	11	16	94	1

As shown in Tab. I, the largest semi-prime we successfully factorized is 744647, which took approximately 47 minutes to compute. Additionally, we observe that as the number of bits in the semi-primes increases, both the number of qubits and the number of quantum gates grow linearly to the number of bits. However, since we are still using a classical simulator to obtain results from the quantum circuits, the processing time increases exponentially.

IV. CONCLUSION

In this Hackathon, we successfully implemented the method proposed in the original paper within **five hours**. Furthermore, we **redesigned the quantum circuit** to improve efficiency by addressing its inherent complexity and inefficiency. Specifically, the original approach relied on a **TableLookup** mechanism to retrieve modular multiplication results. Instead of using lookup tables, we directly implemented **controlled modular multiplication** within the quantum circuit using **CNOT gates**. To obtain results within 24 hours, we further optimized the algorithm using our own techniques by compressing the original design into a version specifically tailored for handling a smaller number of qubits (98-bit instances). Although our implementation runs successfully on a 200-qubit simulator, the original algorithm theoretically supports up to 397-bit instances for RSA-120 factorization. Given sufficient computational resources and runtime, our approach can be extended to match this upper limit.

This optimization significantly reduces the required **workspace register** and **workspace ancilla qubits**, leading to a more resource-efficient implementation. However, due to time constraints, we have not yet conducted a detailed comparison to quantify the improvements in **circuit depth** and **qubit count**. Future work will focus on evaluating these differences and further optimizing the implementation.

Furthermore, we successfully implemented the algorithm on QuantumRing's hardware, validating its feasibility on a real quantum computing platform.

It is particularly noteworthy that none of our team members had prior experience researching **Shor's algorithm**. However, through this MIT Hackathon, we seized the opportunity to learn from the **latest 2024 research** and successfully replicated its implementation within a short timeframe.

What makes this achievement even more remarkable is that we went beyond merely reproducing the original work. We modified the **source code** provided in the paper, transforming it from a specialized implementation designed exclusively for factoring **2048-bit semiprimes** into a more **versatile algorithm** capable of factoring **any semiprime** with adjustable parameters. This enhancement significantly broadens the applicability of the algorithm, making it more **flexible and adaptable** for a wider range of **cryptographic and computational problems**.

[1] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM review **41**, 303 (1999).

[2] S. Beauregard, Circuit for shor's algorithm using 2n+ 3 qubits, arXiv preprint quant-ph/0205095 (2002).

[3] C. Gidney, Factoring with n+ 2 clean qubits and n-1 dirty qubits, arXiv preprint arXiv:1706.07884 (2017).

[4] M. Rossi, L. Asproni, D. Caputo, S. Rossi, A. Cusinato, R. Marini, A. Agosti, and M. Magagnini, Using shor's algorithm on near term quantum computers: a reduced version, Quantum Machine Intelligence **4**, 18 (2022).

[5] É. Gouzien and N. Sangouard, Factoring 2048-bit rsa integers in 177 days with 13 436 qubits and a multimode memory, Physical review letters **127**, 140503 (2021).

[6] H. Arukala, Factorization in cybersecurity: a dual role of defense and vulnerability in the age of quantum computing, (2024).

[7] C. Gidney and M. Ekerå, How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits, Quantum **5**, 433 (2021).

[8] C. Chevignard, P.-A. Fouque, and A. Schrottenloher, Reducing the number of qubits in quantum factoring, Cryptology ePrint Archive, Paper 2024/222 (2024).