

How to factor 2048 bit RSA integers with less than a million noisy qubits

Craig Gidney

Google Quantum AI, Santa Barbara, California 93117, USA

June 9, 2025

Planning the transition to quantum-safe cryptosystems requires understanding the cost of quantum attacks on vulnerable cryptosystems. In Gidney+Ekerå 2019, I co-published an estimate stating that 2048 bit RSA integers could be factored in eight hours by a quantum computer with 20 million noisy qubits. In this paper, I substantially reduce the number of qubits required. I estimate that a 2048 bit RSA integer could be factored in less than a week by a quantum computer with less than a million noisy qubits. I make the same assumptions as in 2019: a square grid of qubits with nearest neighbor connections, a uniform gate error rate of 0.1%, a surface code cycle time of 1 microsecond, and a control system reaction time of 10 microseconds.

The qubit count reduction comes mainly from using approximate residue arithmetic (Chevignard+Fouque+Schrottenloher 2024), from storing idle logical qubits with yoked surface codes (Gidney+Newman+Brooks+Jones 2023), and from allocating less space to magic state distillation by using magic state cultivation (Gidney+Shutty+Jones 2024). The longer runtime is mainly due to performing more Toffoli gates and using fewer magic state factories compared to Gidney+Ekerå 2019. That said, I reduce the Toffoli count by over 100x compared to Chevignard+Fouque+Schrottenloher 2024.

Data availability: *Code and assets created for this paper are available [on Zenodo](#) [[Gid25](#)].*

1	Introduction	3
2	Methods	3
2.1	Approximate Residue Arithmetic	3
2.2	Approximate Period Finding	7
2.3	Ekerå-Håstad Period Finding	13
2.4	Arithmetic Optimizations	14
3	Results	16
3.1	Logical Costs	16
3.2	Physical Costs	18
4	Conclusion	21
5	Acknowledgments	21
A	Detailed Mock-ups	26
A.1	Reference Python Implementation of Algorithm	26
A.2	Addition Operation	30
A.3	Phaseup Operation	34
A.4	Lookup Operation	35
A.5	Frequency Basis Measurement	38

Craig Gidney: craig.gidney@gmail.com

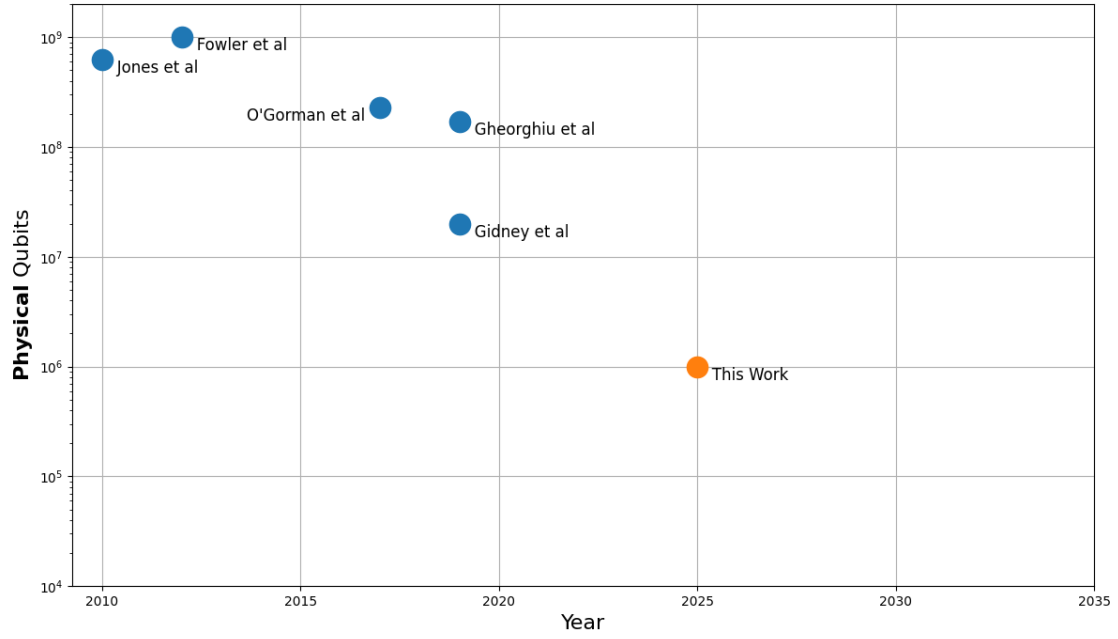


Figure 1: Historical estimates, with comparable physical assumptions, of the physical qubit cost of factoring 2048 bit RSA integers. Includes overheads from fault tolerance, routing, and distillation. Results are from [Jon+12; Fow+12; OC17; GM19; GE21]. Results such as [Van+10] and [LN22] aren't included because they target substantially different assumptions or cost models.

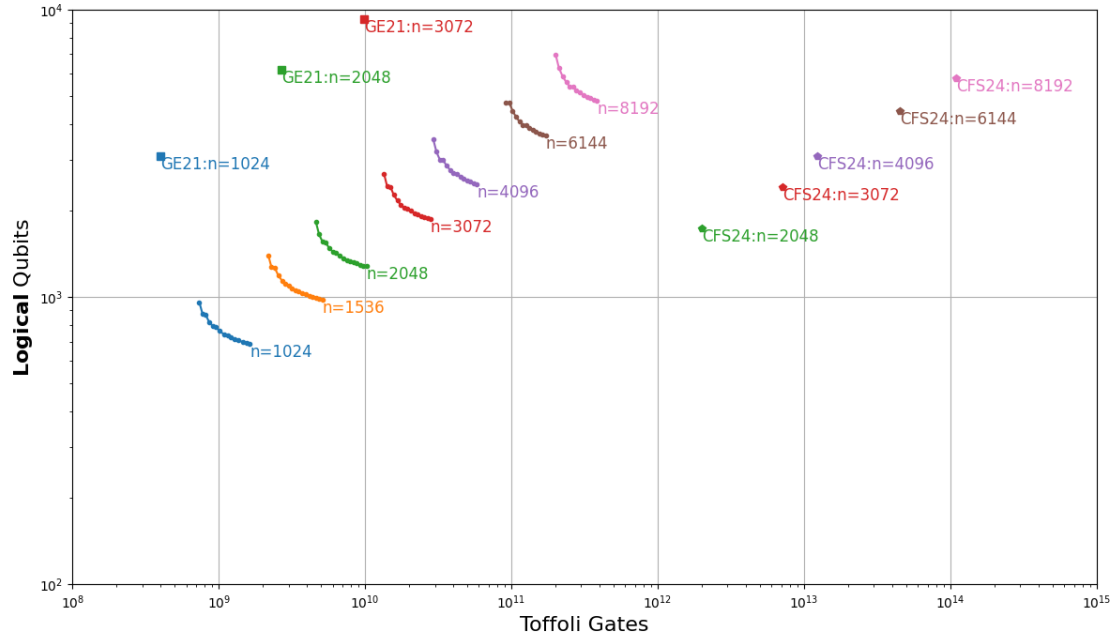


Figure 2: Pareto frontiers achieved by this paper for the Toffoli and logical qubit cost of factoring n bit RSA integers, for various values of n . This paper uses notably fewer logical qubits than [GE21] (points labeled "GE21") and notably fewer Toffolis than [CFS24] (points labeled "CFS24").

1 Introduction

In 1994, Peter Shor published a paper showing quantum computers could efficiently factor integers [Sho94], meaning the RSA cryptosystem [RSA78] wasn't secure against quantum computers. Understanding the cost of quantum factoring is important for planning and coordinating the transition away from RSA, and other cryptosystems vulnerable to quantum computers. Correspondingly, since Shor's paper, substantial effort has gone into understanding the cost of quantum factoring [Kni95; Bec+96; VBE96; Zal98; CW; Bea03; Zal06; Whi+09; Van+10; Fow+12; Jon+12; PG14; Eke16; HRS16; Gid17; OC17; GM19; Eke19; Eke21; GE21; LN22; MS22; Reg24; CFS24] (and more). A key metric is the number of qubits used by the algorithm, since this bounds the required size of quantum computer.

Historically, there was no known way to factor an n bit number using fewer than $1.5n$ logical qubits [Zal06; Bea03; HRS16; Gid17]. Anecdotally, it was widely assumed that n was the minimum possible because doing arithmetic modulo an n bit number "required" an n qubit register. May and Schlieper had shown in 2019 that in principle only a single *output* qubit was needed [MS22], but there was no known way to prepare a relevant output that didn't involve intermediate values spanning n qubits. In 2024, Chevignard and Fouque and Schrottenloher (CFS) solved this problem [CFS24]. They found a way to compute approximate modular exponentiations using only small intermediate values, destroying the anecdotal n qubit arithmetic bottleneck. However, their method is incompatible with "qubit recycling" [PP00; ME99], reviving an old bottleneck on the number of *input* qubits. Shor's original algorithm used $2n$ input qubits [Sho94], but Ekerå and Håstad proved in 2017 that $(0.5 + \epsilon)n$ input qubits was sufficient for RSA integers [EH17]. So, by combining May et al's result with Ekerå et al's result, the CFS algorithm can factor n bit RSA integers using only $(0.5 + \epsilon)n$ logical qubits [CFS24].

A notable downside of [CFS24] was its gate count. In [GE21], it was estimated that a 2048 bit RSA integer could be factored using 3 billion Toffoli gates and a bit more than $3n$ logical qubits. Whereas [CFS24] uses 2 trillion Toffoli gates and a bit more than $0.5n$ logical qubits. So [CFS24] is paying roughly 1000x more Toffolis for a 6x reduction in space. This is a strikingly inefficient spacetime trade-off. However, as I'll show in this paper, the trade-off can be made orders of magnitude more forgiving. In addition to optimizing the Toffoli count of the algorithm, I'll provide a physical cost estimate showing it should be possible to factor 2048 bit RSA integers in less than a week using less than a million physical qubits (under the assumptions mentioned in the abstract).

The paper is structured as follows. In [Section 2](#), I describe a streamlined version of the CFS algorithm. In [Section 3](#), I estimate its cost. I first estimate Toffoli counts and logical qubit counts, and then convert these into physical qubit cost estimates accounting for the overhead of fault tolerance. Finally, in [Section 4](#), I summarize my results. The paper also includes [Appendix A](#), which shows more detailed mock-ups of the algorithm and its physical implementation.

2 Methods

In this section, I present a variation of the CFS algorithm. The underlying ideas are the same, but many of the details are different. For example, I use fewer intermediate values and I extract the most significant bits of the result rather than the least significant bits.

Beware that, as in [GE21], for simplicity, I will describe everything in terms of period finding against $f(e) = g^e \bmod N$ (as in Shor's original algorithm) despite actually intending to use Ekerå-Håstad-style period finding [EH17]. Ultimately everything decomposes into a series of quantum controlled multiplications, which fundamentally is what is actually being optimized, so what I describe will trivially translate.

2.1 Approximate Residue Arithmetic

Consider an integer L equal to the product of many ℓ -bit primes from a set P :

$$L = \prod_{p \in P} p \tag{1}$$