Shor's algorithm compiled on cat qubits: 22 593 qubits for RSA-2048 in 3.14 days

Axel Pappalardo[1], Dahmun Goudarzi[1], Victor Hertz[1], Hugo Jacinto[1,2], Larissa Kroell[1], André Schrottenloher[3], Peter Shanahan[1,4], Élie Gouzien[1], Rémi de La Vieuville[1], and Vivien Londe[1]

[1]Alice & Bob, 49 Bd du Général Martial Valin, 75015 Paris, France
[2]Université Paris–Saclay, CEA, CNRS, Institut de Physique Théorique, 91 191 Gif-sur-Yvette, France
[3]Univ Rennes, Inria, CNRS, IRISA, Rennes, France
[4]School of Informatics, University of Edinburgh, United Kingdom

arXiv:submit/8398585 [quant-ph] 19 Nov 2026

We present an end-to-end resource estimation of Shor's algorithm on superconducting cat qubits, from its mathematical description to the allocation and scheduling of physical qubits on a chip layout. After proposing a comprehensive view of the modern versions of Shor's algorithm for RSA, ECDLP and FFDLP, we make explicit their logical circuits and the trade-offs they enable. We compile the dominant subroutines down to physical-qubit operations on two architectures: one based on repetition cat codes and one based on LDPC cat codes. We design a parity-Cnot gate for cat qubits, extend a multi-target Cnot gate to the LDPC-cat architecture, and adapt an active injection procedure to cat-qubit architectures. Building on these primitives, we adapt to cat-qubit architectures the adder from Gidney's 2025 resource estimate for factoring RSA-2048. We combine our adder compilation and a coarser resource estimate for the phaseup with the factory designs and lookup compilation from the FeMoco resource estimation of Kroell et al. Together, these compilations lead to chip layouts and schedules for factoring RSA-2048.

Assuming a $1\,\mu s$ error-correction cycle time, a $10\,\mu s$ reaction time, circuit-level phase-flip noise with probability $p_Z = 5 \times 10^{-4}$ and no bit-flips, factoring RSA-2048, the configuration we compile in full, requires about 23 000 cat qubits and 3.14 days. Reusing the operations compiled for RSA-2048, solving ECDLP on secp256k1 requires at most 30 000 cat qubits and 1 hour, and FFDLP-512, which we propose as an intermediate benchmark, about 6 700 cat qubits and 21 minutes. Trading space for time, RSA-2048 runs in 1.25 hours on about 611 000 cat qubits. The requirement on no bit-flips (which corresponds to a bit-flip time of about 100 years) can be relaxed to a bit-flip time of about one hour by using an additional concatenation layer into a distance-3 bit-flip error-correcting code. In that case, we estimate that the cost of RSA-2048 increases to about 45 000 cat qubits.

Axel Pappalardo: These authors contributed equally to this work.

Dahmun Goudarzi: These authors contributed equally to this work.

1 Shor's algorithm, a universal benchmark for quantum computing

In 1994, Peter Shor introduced a quantum algorithm to factor a large number or compute a discrete logarithm in polynomial time [1, 2]. The consequences of Shor's algorithm went far beyond the community of cryptographers, as the three main public-key schemes in use at the time were threatened: *RSA* (factorization, with a classical subexponential complexity), *FFDLP* (discrete logarithm on finite field $\mathbb{Z}/p\mathbb{Z}$, which is classically subexponential as well) and *ECDLP* (discrete logarithm on elliptic curves, for which the best known classical attack is exponential). This significant impact on encrypted communications, digital signature, Internet authentication, or more recently crypto-currencies, is one of the reasons the race toward fault-tolerant quantum computing began [3–7].

Since its original publication, the study of Shor's algorithm has led to a dedicated research field, yielding substantial gains in terms of number of logical qubits and duration required to run the algorithm. The avenues for tackling RSA and FFDLP converged into a single algorithm and innovations from Ekerå and Håstad [8], followed by May and Schlieper [9], then by Chevignard et al [10] and Gidney [11], reduced the number of logical qubits from several times the size of the key to half of it. The development of the algorithm for ECDLP is just as dense, with three different paths of optimization emerging in recent publications [12–14]. This work relies on the modern version of these algorithms to solve RSA, FFDLP, and ECDLP (see Appendix A for a comprehensive presentation and a step-by-step example).

Apart from its practical impact, Shor's algorithm applied to RSA is also a perfect candidate for benchmarking and comparing the architectures of quantum computers, for three reasons. First, no known classical algorithm can factor integers of cryptographic size (e.g. RSA keys of more than 2048 bits) in a reasonable time. A successful run of Shor's algorithm at such sizes is therefore strong evidence that the computation was run on a quantum computer. Second, the problem it solves is simple to state, and the results it provides are easy to verify (e.g. checking that the product of two numbers is equal to a public key). Finally, due to its required depth, the computation cannot succeed without large-scale quantum error correction. In fact, to run the algorithm, billions of logical operations need to be executed without errors, several orders of magnitude beyond the thousands of operations that a machine without error correction can execute.

Benchmarking on RSA-2048 and beyond is useful for comparing architectures, but will only become practical in a few years. Thus, we introduce here an additional intermediate benchmark, *FFDLP-512*, defined as an instance of the FFDLP problem for a 512-bit prime group and an 80-bit exponent. Although the cryptanalysis of this problem can be performed classically, it offers three advantages that make it relevant as a first cryptographic milestone.

First, Shor's algorithm for FFDLP is space-efficient. The size of the group targeted in FFDLP-512 makes this intermediate benchmark practical significantly earlier than the RSA-2048 benchmark.

Second, the implementation uses all the blocks needed for RSA-2048, hence it can demonstrate an actual end-to-end execution of Shor's algorithm without shortcuts, contrary to toy problems such as factoring 128-bit numbers where some of the building blocks of the algorithm become irrelevant.

Finally, such Diffie–Hellman parameters have been observed at a large scale on production systems. A scan performed in 2015 by the authors of the Logjam attack [15] revealed that 8.4% of the top 1 million domains were accepting 512-bit groups ("export grade"), with two groups covering 92% of such vulnerable servers.

2 Measuring the cost of a quantum computation

Formally comparing two descriptions of a quantum computer (with different qubit modalities, architecture and/or error correction scheme) requires agreement on metrics to take into account, in the context of the specific application considered for the benchmark is defined (Shor's algorithm applied respectively to RSA-2048 challenge number, Bitcoin curve (secp256k1), and FFDLP-512 for an intermediate benchmark).

The two primary metrics used across publications to compare architectures are *number of physical qubits* (depending on the qubit modality, the error correction architecture and the precise compilation of the algorithm to a specific architecture) and *total execution time*. Space-time trade-offs between number of physical qubits and execution time are generally possible. Other metrics are important, but are more applicable to assessing the feasibility of an architecture (e.g. cost, footprint, energy consumption...).

Three rather independent levers enable the improvement of these two main metrics, and explain the rapid evolution of Shor's algorithm from 1994 to 2026:

- *The high-level algorithm:* reducing the number of logical qubits, the number of costly gates (typically Toffoli or T gates in most fault-tolerant architectures) or improving success probability;
- *Hardware architecture:* improving individual physical qubits (faster, more stable, or with a

noise profile favorable to error correction), physical operations (single- and multi-qubit gates and measurements), qubit arrangement in the system (connectivity, modularity, routing), and improvements on fabrication and enabling technologies;

- *Quantum Error Correction (QEC):* better logical-to-physical qubit ratio, faster error correction cycles, higher fidelity of costly gates, etc.

Each publication improving Shor's algorithm performance acts on one to three of these levers, defining a triplet: a specific algorithm, targeting a specific hardware platform, compiling the algorithm to the architecture through specific QEC implementation [16, 17].

3 Superconducting cat qubits

The qubit technology considered in this work is the dissipative cat qubit [18–21] (a variant, the Kerr-cat qubit, is stabilized by a two-photon drive instead [22–24]), which belongs to the family of superconducting qubits, i.e. small electronic circuits exhibiting quantum properties. The main difference from transmons (superconducting modality assumed in Gidney–Ekerå [25]) is that instead of encoding the state in two levels of energy of an anharmonic oscillator, the cat qubit circuit is engineered so that the state is encoded in superpositions of the two coherent states of opposite phase ("Schrödinger cat states") of an electromagnetic mode.

This specific encoding has very practical consequences. Qubits are typically subject to two kinds of noise, *bit-flip* (an unwanted flip of $|0\rangle$ into $|1\rangle$ and vice versa, as for classical bits) and *phase-flip* (from $|+\rangle$ to $|-\rangle$, changing the phase of the qubit). In practice, the errors affecting a qubit can essentially be described as a combination of both bit-flips and phase-flips, so correcting these two types is enough to protect it.

For cat qubits, the bit-flip rate decreases exponentially with a tunable parameter (the number of photons in the cat state), while the phase-flip rate increases only linearly with this parameter [26–31].

This bias between the two types of noise enables simpler error correction schemes (such as a repetition code architecture [32–36]), with a much lower logical-to-physical qubit ratio, as only phase-flips need to be corrected. More advanced error correction codes, such as LDPC (low-density parity-check) codes, further lower this ratio [16]. For the deepest computations, such as Shor's algorithm for RSA-2048, some bit-flip correction beyond the intrinsic protection of cat qubits might be required [16], and an architecture based on elevator codes has been proposed to do this [37]. Lower-depth computations such as FFDLP-512 require a shorter bit-flip time, of about two months instead of about a hundred years for RSA-2048, which nevertheless remains far beyond the bit-flip times measured to date [27, 29, 31] (Section C.9). In the context of this paper, the noise model of a cat qubit can be summarized with a single parameter p_Z (phase-flip probability), which is mostly driven by the ratio between κ_2 (two-photon dissipation, which is the effect that stabilizes the cat state) and κ_1 (one-photon loss, the effect that degrades the cat state) (Section C.9.1), a critical hardware feature. Bit flips are assumed to be negligible, and Section C.9.3 quantifies the corresponding requirement on the bit-flip time.

4 Abstraction layers and compilation

A fault-tolerant quantum algorithm is compiled through several layers of abstraction down to instructions for a specific machine.

The highest level of abstraction is the *Algorithm layer*: in the context of this work, a description of Shor's algorithm and its subroutines (modular exponentiation, Quantum Fourier Transform...). Algorithms at this level are described with block diagrams, pseudo-code and mathematical representation of the transformations applied to registers.[1]

The algorithm is then compiled to a *Logical layer*. This is the lowest level of abstraction that is still independent of the quantum technology considered, thus enabling hardware-agnostic comparisons between algorithms. Each subroutine is described precisely in terms of logical qubits and logical gates (e.g. CNOT gate, Toffoli gate, ...) or logical operations (e.g. adder, lookup, ...). A basic resource estimation is possible at this level, using approximate metrics such as the logical-to-physical qubit ratio of an architecture [39]. However, such approaches do not accurately account for the impact of routing qubits on space and time estimates. In this work, logical circuits are implemented in Qualtran and Qarton, two Python libraries [40, 41].

The lowest layer of abstraction considered here is the *Physical layer*, which is specific to the targeted quantum hardware. This level specifies the allocation of each physical qubit, or group of physical qubits, to a function (data logical qubit, routing logical qubit, ancilla, magic state factory), through a dynamic mapping between logical and physical qubits, possibly with a different error-correcting code depending on the operations applied to a logical qubit or the duration for which its state must be stored. Optimizations of the compilation can go directly from operation specifications (e.g. an adder description) to this physical level, without going through the logical circuit abstraction.

[1] We recommend reading [38] to gain familiarity with qubits, gates and Dirac notation $|\cdot\rangle$ for quantum states.

With this last layer, we can precisely evaluate the number of physical qubits used at each time step, the duration of each subroutine, and the number and type of magic state factories, and deduce the total number of qubits and duration. This analysis enables the comparison of different hardware platforms, but also the design of other parts of the system such as the throughput of the decoders.

In this work, we propose architectures where each cat qubit is allocated to a specific zone, and their use is scheduled at every time step.

In the repetition-cat architecture, every logical qubit is a repetition code of distance d (the size of the code, which sets its level of protection).

In the LDPC-cat architecture (Figure 1), the chip is a flip chip made of two facing layers. An LDPC layer stores the registers of the algorithm in a single, dense LDPC-cat code block of distance d_{LDPC}. A repetition layer underneath gives access to these registers and holds the routing qubits, some auxiliary qubits and buffers that store the outputs of distillation factories in repetition codes of distance d_{rep}.

5 Contributions and main results

This work gives an end-to-end resource estimation of Shor's algorithm, from its mathematical description to the allocation and scheduling of physical qubits on a cat-qubit architecture. The contributions follow the structure of the abstraction layers introduced in Section 4.

A reference description of modern Shor. Since 1994, improvements to Shor's algorithm have appeared in many publications, improving a specific part of the algorithm, or targeting only one cryptographic problem (RSA, ECDLP or FFDLP). Appendix A presents these improvements in a unified way, with examples making it usable as a comprehensive introduction for readers outside of the quantum community.

For RSA and FFDLP, four optimizations are presented: reduction to a short discrete logarithm problem (Ekerå–Håstad), shorter input register (May–Schlieper), residue arithmetic and approximate period finding (Chevignard–Fouque–Schrottenloher, then Gidney). We derive the failure budget of the algorithm, and provide the expected number of shots and the algorithmic error rate. We also introduce the notion of noisy hidden subgroup problem in order to formalize the introduction of approximate arithmetic induced by the latest optimizations.

An alternative approach to Shor's algorithm exists for factoring, introduced by Regev in 2023 [42] and further optimized in various works [43–45]. Nonetheless, the current best optimizations of Regev's algorithm are still less practical than the latest optimizations of Shor's algorithm. For ECDLP, there is no single best algorithm in space and time. We compare gate-optimized, compressed and register-sharing constructions, and identify the resulting Pareto frontier.

An explicit logical implementation. Appendix B describes all logical circuits explicitly, and their implementation in the Qarton and Qualtran libraries. The circuit uses gates that have a classical analog (X, CX, CCX), phase gates and measurement-based uncomputation, providing exact logical qubit and Toffoli counts. This section enables the extraction, for each algorithm, of the *profile* of its implementation: time and space of each subroutine, share of the overall qubit usage, and percentage of the total runtime. Given a set of physical instructions and an architecture, the tables derived for RSA/FFDLP and ECDLP allow to compile the algorithm and output an estimate of the required number of physical qubits and runtime. We also propose the first implementation of FFDLP for different sets of parameters in Qarton and Qualtran.

Physical compilation on cat qubits. Most of the novelty of this work lies at the Physical layer (Appendix C). Instead of multiplying logical qubit numbers by an average logical-to-physical ratio, we compile the dominant subroutines directly to physical qubits.

We design a *parity-*CNOT *gate* for cat qubits, which runs in d error-correction cycles regardless of the number of controls, extend the *multi-target* CNOT *gate* to the LDPC-cat architecture, and adapt from Gidney [11] a procedure we call *active injection*, which applies a Toffoli gate to parities of qubits in the time of a standard injection.

We build on these primitives to redesign the adder, the most frequently called operation. The construction of [11] assumes surface code costs, in which $|i\rangle$ states and Hadamard gates are cheap. On a cat-qubit architecture, both are expensive, so we rebuild the adder around auto-AND and delayed-C$_X X$ resource states. Simultaneous lattice surgery [46] and pipelined resource-state preparation let the adder keep pace with a single magic-state factory, as long as the code distance stays below the 21 error-correction cycles the factory takes per magic state. For both the repetition-cat and the LDPC-cat architectures, we give the full layout and schedule of the adder, together with the sizing of the factories that feed it. Combining this adder with the factory designs and the lookup compilation of [47], and with a coarser estimate for phase-ups, we obtain global chip layouts and schedules, and from them a count of physical qubits and of the execution time.

For ECDLP, we used the operations we optimized for RSA (adder and lookup) and applied the profiling

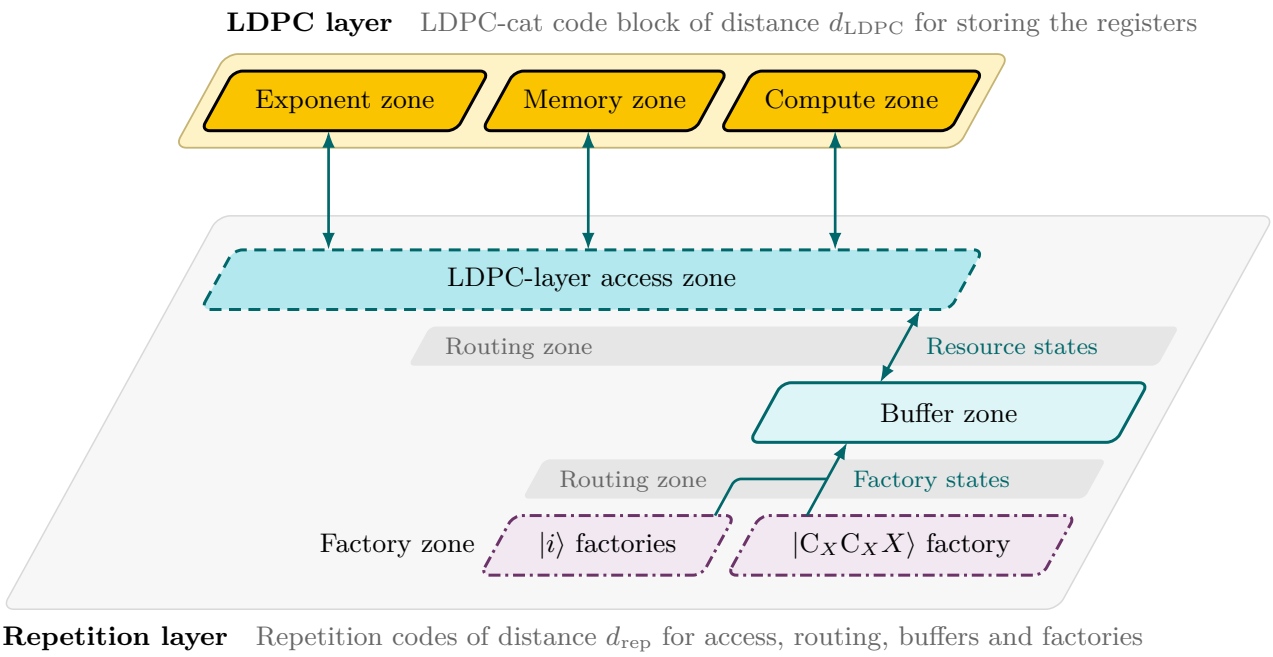


Figure 1: Simplified global layout of the LDPC-cat architecture for RSA-2048 as presented in Figure 53 of Appendix C.

table of Appendix B to obtain rough resource estimates. This is not an optimal path as most of the subroutines for ECDLP and its physical layout could be further optimized. Nevertheless, it allows us to derive rough upper bounds on the number of physical qubits and runtime required for this problem.

Resource estimations. Table 1 summarizes the results for the three instances. We assume an error-correction cycle of 1 µs, a decoder reaction time of 10 µs, a phase-flip circuit-level noise model with probability $p_Z = 5 \times 10^{-4}$, and no bit flips during the computation. In practice, this requires a bit-flip time of at least about 100 years, for which the bit flips lengthen the runtime by about 11 %; for a bit-flip time ten times longer, this overhead becomes negligible (Section C.9.3). Under these assumptions:

- factoring a 2048-bit RSA modulus requires 22 593 cat qubits and 3.14 days;
- solving ECDLP on the Bitcoin curve (secp256k1) requires about 29 588 cat qubits and 1 hour;
- the FFDLP-512 benchmark requires about 6 656 cat qubits and 21 minutes.

The RSA-2048 configuration is the main focus of this work: it is the only one compiled in full at the physical level (Appendix C), and its figures are the most precise. The other results (Appendix D) are derived from it. The ECDLP figures reuse the RSA operations and are therefore upper bounds, and the FFDLP-512 figures reuse the same compiled operations with the parameters of Table 5.

Possible space-time trade-offs are presented in Table 2. With more qubits, RSA-2048 runs in about one hour on 610 530 cat qubits. With fewer qubits, RSA-2048 needs 21 188 cat qubits over 8 days. These trade-offs are extrapolated from the balanced configuration. We now discuss how this figure changes if we degrade the physical assumptions or the architecture. With a phase-flip probability $p_Z = 10^{-3}$ instead of 5×10^{-4}, factoring RSA-2048 requires 25 673 cat qubits and 3.63 days (Table 29). If the bit-flip time is only of the order of one hour, an additional layer of bit-flip protection is needed, and we estimate that the cost increases to about 45 000 cat qubits and at least 3.14 days (Section C.9.4). Finally, without LDPC-cat codes, the repetition-cat architecture requires 49 649 cat qubits and 2.67 days at $p_Z = 5 \times 10^{-4}$ (Section C.7.2).

6 Conclusion

Figure 2 presents a comparison of the results of this work with previous publications since 2010. We observe that in fifteen years, progress on algorithms, error correction and hardware architecture cut the space-time volume by 5 orders of magnitude, from 10^4 Mqubit.day for initial surface-code estimations to 7×10^{-2} Mqubit.day. With different hardware hypotheses, this work offers a 40x reduction of the physical qubit count compared to [11], for a shorter runtime. Hence, these results position cat qubits as both a hardware-efficient and time-efficient platform.

From a practical standpoint, the reduction of the size of a quantum computer running Shor's algorithm

Instance	Logical qubits	Toffoli / shot	Physical qubits	Time / shot	Shots	Total time
RSA-2048	1 433	$2^{29.5}$	22 593	7.84 h	9.60	3.14 days
ECDLP-256	1 462	$2^{25.8}$	29 588	60 min	1.03	61.8 min
FFDLP-512	208	$2^{22.2}$	6 656	2.51 min	8.44	21.2 min

Table 1: Logical and physical resources for the three target instances (balanced configuration, LDPC-cat architecture, $p_Z = 5 \times 10^{-4}$, negligible bit flips, see Section C.9.3). Space-time trade-offs are given in Table 2.

	RSA-2048		ECDLP-256		FFDLP-512	
	Nb cat qubits	Duration	Nb cat qubits	Duration	Nb cat qubits	Duration
Balanced	22 593	3.14 days	29 588	61.8 minutes	6 656	21.2 minutes
Space-efficient	21 188	8.08 days	28 183	146 minutes	5 251	58.5 minutes
Time-efficient	610 530	1.25 hours	-	-	66 560	2.6 minutes

Table 2: Trade-off between physical qubit count and runtime for the three targeted problems, on the LDPC-cat architecture.

for RSA-2048 increases the need for a fast transition to post-quantum cryptography. Our results push for an immediate migration effort, relying on hybrid schemes during the transition phase.

Results on ECDLP have practical consequences for the security of Bitcoin cryptocurrency. Space-efficient configuration is a threat to exposed public keys, such as P2PK outputs or reused addresses [6].

The FFDLP-512 milestone proposed in this work enables to run a quantum computation based on Shor's algorithm with far fewer qubits. We recommend adopting FFDLP-512 and RSA-2048 as common benchmarks for all fault-tolerant architectures.

All the algorithms and subroutines discussed here are still an active object of research, and leave room for improvements over the coming years. However, we want to highlight the importance of also studying the primitives that will replace RSA and elliptic curves. The trust we place in them for post-quantum cryptography relies on the lack of efficient quantum cryptanalysis. Thus, the same level of effort should be put toward the study of algorithms with potential application to these new primitives. Maturity of quantum cryptanalysis is a precondition for trust in the cryptography intended to resist it.

7 Acknowledgments and author contributions

The authors would like to thank André Chailloux and Antoine Joux for their careful review of preliminary versions of this work; Timo Achard for his expertise in Qualtran; and Christelle Yammine for her attention to style coherence and accessibility. We also thank Jérémie Guillaud, Christophe Vuillot and Wes Roberts for their careful proofreading of the manuscript and Laurent Prost for insightful discussions on Shor's algorithm.

PS acknowledges EPSRC CDT grant EP/Y035097/1. A.S. acknowledges support by the French Agence Nationale de la Recherche through the France 2030 program under grant agreement No. ANR-22-PETQ-0008 PQ-TLS. A.S.'s work was done in part while visiting the Simons Institute for the Theory of Computing, supported by a grant from the National Science Foundation.

DG wrote the algorithm overview in Appendix A and contributed to Appendices B and D. AP wrote the logical implementation in Appendix B for RSA/FFDLP and parts of the resource estimation in Appendices C and D. VH wrote the adder implementation in Section C.5 and contributed to the rest of the resource estimation in Appendix C. HJ and LK contributed to the resource estimation in Appendix C. AS wrote the logical implementation in Appendix B for ECDLP and wrote the different Qarton implementations. PS estimated the Elevator code overhead in Section C.9.4. EG contributed to the algorithm overview and its logical implementation and supervised preliminary work on injection gadgets. AP and DG wrote the Qualtran implementation. RLV managed the project and wrote the main text. VL supervised and contributed to the logical implementation and the resource estimation in Appendices B and C.

Claude Opus 5.5 (Anthropic) [48] was used to review the text of this work and our Qualtran implementations, help rewrite some sections, and generate some figures. The authors checked all of its outputs and take full responsibility for the content of this work.

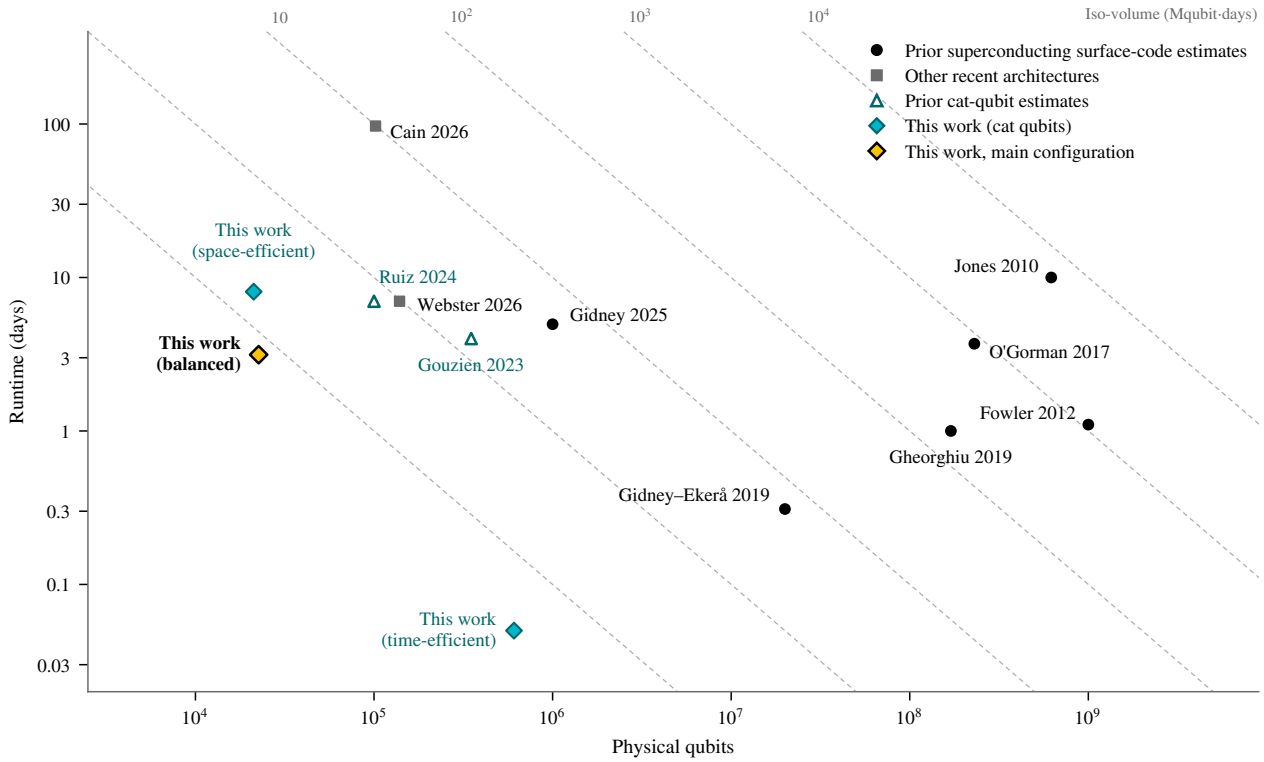


Figure 2: Physical qubit count versus runtime for factoring RSA-2048, on logarithmic axes. Dashed diagonals are lines of constant spacetime volume (qubits $\times$ days), labelled in megaqubit·days along the top axis.

Contents

A Deep Dive into Shor's Algorithm **9**
A.0 High-level Overview 9
A.1 Hidden Subgroup Problem 11
Factoring. 11
Discrete logarithm. 12
A.2 The Noisy Hidden Subgroup Problem 12
Four relaxations of the exact promise. 13
A.3 Shor's Algorithm 14
State preparation. 14
Oracle evaluation. 14
Quantum Fourier transform. . 14
Classical post-processing 17
Efficient implementation of the oracle. 19
Overall cost. 20
A.4 From Factoring to Short Discrete Logarithm Problem 21
A.5 Reducing the Input Register Size . . . 22
A.6 Reducing the Output Register Size . . 25
A.7 Making the Oracle Function Space Efficient 26
A.8 Approximate Period Finding 30
A.9 Assembling Modern Shor 34
A.10 FFDLP Algorithm 38
A.11 Parameter Selection for RSA and FFDLP 39
A.12 The Elliptic-Curve Instance 39
A.13 Optimize the Gates: Splitting the EEA 45
A.14 Shrink the Register: Sharing the EEA Exactly 46
A.15 Avoid the Inversion: Projective Coordinates and a One-Bit Output 48
A.16 Trade-offs for ECDLP 48

B Logical Implementation of Shor's Algorithm **51**
B.1 Approximate Modular Exponentiation for Factoring 51
B.1.1 Computing the logarithm of the residue 53
B.1.2 Reducing the exponent 53
B.1.3 Computing the residue and its uncomputation 55
B.1.4 Accumulating the residue . . . 58
B.1.5 Logical cost breakdown 58
B.2 Arithmetic circuits for the two ECDLP variants 59
B.2.1 High-level point addition . . . 59
B.2.2 Out-of-place squaring 60
B.2.3 In-place modular inverse using dialogs 60
B.2.4 Additive modular operations . 63
B.2.5 Logical cost breakdown 63

C Resource estimation for RSA-2048 **66**
C.1 Architecture 66
C.1.1 Layout, connectivity and noise 66
C.1.2 Zones 66
C.1.3 Error correcting codes 66
C.2 Lattice-surgery primitives 67
C.2.1 Parity-CNOT gate on cat qubits 67
C.2.2 Multi-target CNOT gate on cat qubits 67
C.2.3 Active injection 69
C.3 Color-changed implementation 69
C.4 Magic-state factories and cadence . . . 69
C.5 ADDER 73
C.5.1 Logical circuit 73
C.5.2 Layout on the repetition-cat architecture 74
C.5.3 Layout on the LDPC-cat architecture 74
C.5.4 Resource state preparations . . 75
C.5.5 Simultaneous lattice surgery . . 76
C.5.6 Schedule 77
C.5.7 Summary 79
C.6 LOOKUP and PHASEUP 81
C.7 Repetition-cat architecture 84
C.7.1 Layout, duration and volume at distance d 84
C.7.2 Evaluation at $p_Z = 10^{-3}$ and $p_Z = 5 \times 10^{-4}$ 91
C.8 LDPC-cat architecture 93
C.8.1 Layout, duration and volume at distances d_{rep} and d_{LDPC} . . . 93
C.8.2 Evaluation at $p_Z = 5 \times 10^{-4}$ and $p_Z = 10^{-3}$ 100
C.9 Physical cat qubits and bit-flip assumptions 102
C.9.1 Cat qubits: encoding, stabilization, operations, and bit flips . 102
C.9.2 Experimental state of the art . 104
C.9.3 Comparison with the bit-flip assumptions of the main text . . 106
C.9.4 Additional layer of bit-flip protection 106

D Estimation for the other variations and space-time trade-offs **109**
D.1 RSA 109
D.1.1 Space-efficient configuration . . 109
D.1.2 Time-efficient configuration . . 109
D.2 FFDLP 114
D.2.1 Balanced configuration 114
D.2.2 Space-efficient configuration . . 115
D.2.3 Time-efficient configuration . . 116
D.3 ECDLP 117
D.3.1 Balanced configuration 117
D.3.2 Space-efficient configuration . . 121
D.3.3 Time-efficient configuration . . 122

References **123**

A Deep Dive into Shor's Algorithm

Shor's algorithm [2] is the central quantum threat to classical public-key cryptography. It solves the integer factoring problem and the discrete logarithm problem in polynomial time on a quantum computer. This breaks RSA, Diffie–Hellman, and elliptic-curve-based schemes. This section develops both the original algorithm and its modern descendants for factoring: Ekerå–Håstad [8], May–Schlieper [9], and CFS/Gidney [10, 11]. Together, the most recent algorithms reduce the logical qubit count of attacking RSA from several times the key size to asymptotically half of it. For RSA-2048 this means about 1400 logical qubits [11, Table 5]. We begin with the factoring instance, which serves as a high-level sketch of the algorithm.

A.0 High-level Overview

We start by informally explaining how Shor's algorithm works to recover, from an RSA modulus N, the two factors p and q.

i Classical pre-processing. One first picks a random integer $1 < g < N$ and computes $b = \gcd(g, N)$ using the Euclidean algorithm. If $b \neq 1$, then b is already a non-trivial factor of N, and the algorithm terminates immediately. This case is rare in practice: it would require g to be a multiple of p or q, which happens only with negligible probability $O(\frac{1}{p} + \frac{1}{q})$. If $b = 1$, then $g \in \mathbb{Z}_N^*$, meaning g is invertible modulo N. In fact, since $\gcd(g, N) = 1$, Bézout's identity gives two integers u, v such that $ug + vN = 1$. Reducing mod N yields $ug \equiv 1 \pmod N$, confirming that g is invertible. The powers $g^1, g^2, g^3, \ldots$ can only take finitely many values modulo N. Two of them must eventually repeat: $g^i \equiv g^j \pmod N$ for some $i < j$. Multiplying both sides by g^{-i} gives $g^{j-i} \equiv 1 \pmod N$. So $r := j - i$ is a positive integer with $g^r \equiv 1 \pmod N$

The algorithm proceeds to find the multiplicative order r of g modulo N, i.e. the smallest integer $r > 0$ such that $g^r \equiv 1 \pmod N$.

ii Classical post-processing. At this stage, the order r is known thanks to a quantum oracle (see hereafter). We now need to recover p and q, the two non-trivial factors of N. Assuming two conditions hold, this can be done classically with probability $\geq \frac{1}{2}$. Specifically, suppose that (i) r is even and (ii) $g^{r/2} \not\equiv -1 \bmod N$. Then $g^r - 1$ can be factored as

$$g^r - 1 = (g^{r/2} - 1)(g^{r/2} + 1). \tag{1}$$

By definition of the order, together with the two assumptions, we obtain:

(a) $N \mid g^r - 1$

(b) $g^{r/2} \not\equiv \pm 1 \bmod N$

Therefore, N divides the product of Equation (1) but neither factor. As a result, $\gcd(g^{r/2} - 1, N)$ and $\gcd(g^{r/2} + 1, N)$ are both non-trivial factors of N. Conditions (i) and (ii) hold with probability at least $\frac{1}{2}$ for a random g[2]. Hence, only a constant number of repetitions, with independently chosen g, suffices to recover p and q successfully.

iii Processing: quantum period finding. Only one step can be accelerated quantumly. It is the central one, and it computes the order r. Classically this is expensive. The best known methods take time growing sub-exponentially in the number of bits of N. Let $f\colon e \mapsto g^e \bmod N$ be the function that evaluates all the exponentiation of g modulo N. f is a periodic function of period r, and recovering r from queries to f is an expensive process. A quantum computer instead evaluates f on a superposition of all inputs at once. Because f is periodic with period r, applying the quantum Fourier transform (QFT) concentrates the amplitude on frequencies that are multiples of $1/r$. Section A.3 explains this in detail. Measuring then returns such a multiple, from which r can often be recovered. Running the circuit and measuring the input register yields an integer $c \approx \mu 2^{n_e}/r$. Here n_e is the input register size, and μ is sampled by the measurement. The order r is recovered only when $\gcd(\mu, r) = 1$, which happens with probability $\varphi(r)/r$. Here φ is Euler's totient function, which counts the positive integers up to its argument that are coprime to it. Since $\varphi(r)/r = \Omega(1/\log\log r)$, $O(\log\log r)$ repetitions of the circuit suffice in expectation (Section A.3).

The register holding the input exponents must be large enough for this to work. A QFT over a register of n_e qubits resolves a frequency to a precision of about $1/2^{n_e}$. Separating neighboring multiples of $1/r$ would only need $2^{n_e} \gtrsim r$. Recovering r needs more. The measured fraction $c/2^{n_e}$ must lie within $1/(2r^2)$ of μ/r, so that continued fractions return μ/r in lowest terms (Section A.3). This requires $2^{n_e} \gtrsim r^2$, i.e. about $2n$ qubits for an n-bit modulus. Between

[2] Let p, q be distinct odd primes. By the Chinese remainder theorem a uniform $g \in \mathbb{Z}_N^*$ corresponds to an independent uniform pair $(g_p, g_q) \in \mathbb{Z}_p^* \times \mathbb{Z}_q^*$, with $r = \operatorname{lcm}(\operatorname{ord}_p g_p, \operatorname{ord}_q g_q)$. Let c_p, c_q be the exponents of 2 in $\operatorname{ord}_p g_p$ and $\operatorname{ord}_q g_q$. The two failure modes are exactly $c_p = c_q$. The order r is odd iff $c_p = c_q = 0$, and $g^{r/2} \equiv -1$ iff $c_p = c_q \geq 1$. It therefore suffices to show $\Pr[c_p = c_q] \leq \frac{1}{2}$. Write $p - 1 = 2^{e_p} m_p$ with m_p odd, and write $g_p = \gamma^k$ for a generator γ of the cyclic group $\mathbb{Z}_p^*$, with k uniform in $\{0, \ldots, p-2\}$. Then $\operatorname{ord}_p g_p = (p-1)/\gcd(k, p-1)$, so $c_p = e_p$ exactly when k is odd. This happens with probability $\frac{1}{2}$, hence $\Pr[c_p = c] \leq \frac{1}{2}$ for every c. Since g_p and g_q are independent, $\Pr[c_p = c_q] = \sum_c \Pr[c_q = c] \Pr[c_p = c] \leq \frac{1}{2} \sum_c \Pr[c_q = c] = \frac{1}{2}$. The bound is tight, for example when $p \equiv q \equiv 3 \pmod 4$. Drawing $1 < g < N$ excludes $g = 1$, which fails, so it can only raise the success probability. See [38, Theorem A4.13] for N with more prime factors.

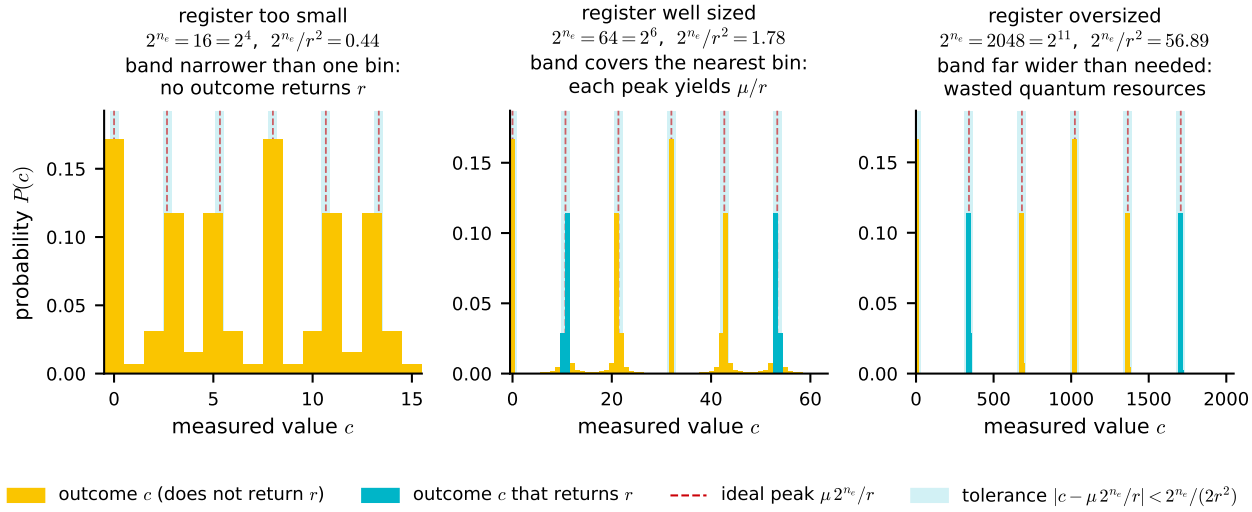


Figure 3: Register size sets peak resolution, for period $r = 6$. Each panel shows the probability $P(c)$ of measuring the outcome c on the n_e-qubit exponent register after the inverse QFT. Dashes mark the ideal peaks $\mu\,2^{n_e}/r$, with $\mu \in \{0, \ldots, r-1\}$. Continued fractions return μ/r whenever c falls in the shaded band $|c - \mu\,2^{n_e}/r| < 2^{n_e}/(2r^2)$ around a peak, and teal bars are the outcomes that return r. Left: $2^{n_e} = 16 < r^2$, so each band is narrower than one bin and no outcome returns r. The two bins inside a band, $c = 0$ and $c = 8$, belong to $\mu = 0$ and $\mu = 3$, which share a factor with r and return only a divisor of it. Middle: $2^{n_e} \gtrsim r^2$, so the band always covers the nearest bin. Right: the register is far larger than needed.

r and r^2 the peaks are still separated, but continued fractions can no longer identify r. Beyond r^2 the arithmetic on the register only grows more costly for no benefit. We depict the register size impact in Figure 3. Crucially, the whole routine (one evaluation of f in superposition, one QFT, and one measurement), repeated $O(\log \log r)$ times in expectation, uses only $\mathrm{poly}(n)$ operations. This is the superpolynomial speedup at the heart of Shor's algorithm: order finding, classically intractable, becomes efficient with a quantum algorithm.

A concrete run. Take $N = 77$. Suppose the pre-processing draws $g = 18$. Then $\gcd(18, 77) = 1$, so we proceed to find the order. Computing successive powers modulo 77,

$$18^1 = 18, \quad 18^2 = 16, \quad 18^3 = 57, \ \ldots,$$
$$18^{30} \equiv 1 \pmod{77},$$

the first return to 1 happens at $r = 30$, so the order of 18 is $r = 30$. This r is even, and $18^{r/2} = 18^{15} \equiv 43 \pmod{77}$, which is neither 1 nor $-1 \equiv 76$. Post-processing then gives

$$\gcd(18^{15} - 1,\, 77) = \gcd(42, 77) = 7,$$
$$\gcd(18^{15} + 1,\, 77) = \gcd(44, 77) = 11,$$

and indeed $77 = 7 \times 11$. Not every base works. Had the pre-processing drawn $g = 15$ instead, its powers are

$$15^1 = 15, \quad 15^2 = 71, \quad 15^3 = 64,$$
$$15^4 = 36, \quad 15^5 \equiv 1 \pmod{77},$$

so the order is $r = 5$, which is *odd*. The identity $g^r - 1 = (g^{r/2} - 1)(g^{r/2} + 1)$ requires an integer exponent $r/2$, so it does not apply here. This run therefore yields no factor. One simply discards $g = 15$ and draws a fresh base. This is the "probability $\geq \frac{1}{2}$" from the post-processing above, so a handful of tries succeeds with overwhelming probability.

Why a period looks like a hidden subgroup. The quantum period-finding step is the heart of the algorithm. The structure the quantum step actually exploits is best seen through *cosets*. Take the integers $\mathbb{Z}$ under addition and the subgroup $H = 2\mathbb{Z}$ of even numbers. Translating H by a fixed element a gives the set $a+H = \{a+h : h \in H\}$, called a coset. Here there are exactly two: $0+H$, the even numbers, and $1+H$, the odd numbers. They *partition* $\mathbb{Z}$: every integer lies in exactly one, and two integers land in the same coset precisely when they differ by an element of H, i.e. by an even number.

Order finding has the same shape. Fix g and consider the function $f(x) = g^x \bmod N$ on the exponents $x \in \mathbb{Z}$. Because $g^r \equiv 1$, we have $f(x + r) = f(x)$: the function repeats with period r. Playing the role of H is the subgroup $r\mathbb{Z} = \{0, \pm r, \pm 2r, \ldots\}$ of exponents that leave f unchanged, and its cosets $a + r\mathbb{Z}$ are the sets of exponents sharing a common value of f. Two properties, which are what the quantum step relies on, hold:

- f is *constant on each coset*: if x and x' differ by a multiple of r, then $f(x) = f(x')$.
- f *separates distinct cosets*: if x and x' lie in different cosets, then $f(x) \neq f(x')$.

Concretely, for $g = 18$, $N = 77$ above, $r = 30$, so the exponents split into 30 cosets $\{0 + 30\mathbb{Z},\ 1 + 30\mathbb{Z},\ \ldots,\ 29 + 30\mathbb{Z}\}$. The coset $1 + 30\mathbb{Z}$, for instance, is exactly the set of exponents x with $18^x \equiv 18 \pmod{77}$. The values $x = 1, 31, 61, \ldots$ all give 18, and no exponent outside this coset does. Recovering the period r is therefore the same task as recovering the hidden subgroup H. The function available is constant on the cosets of H and distinct across them. This is the *Hidden Subgroup Problem*, and stating it precisely is what the next subsection does.

In his seminal paper, Shor [2] also provides a simple extension of this algorithm to the discrete logarithm problem. The extension is natural: both factoring and the discrete logarithm reduce to the same underlying computational primitive, finding a hidden subgroup of an abelian group, and the quantum subroutine is structurally identical in both cases. The only differences lie in the classical pre- and post-processing. We make both reductions explicit in the following subsection.

A.1 Hidden Subgroup Problem

Let us first start by defining what a coset is and the notion of constant and separating for a coset.

Definition A.1 (Coset)**.** Let G be an abelian group and H a subgroup of G. The *coset* of H associated with an element $g \in G$ is the set

$$gH = \{gh : h \in H\}.$$

When G is finite, the cosets of H partition G into $|G|/|H|$ subsets, each of size $|H|$.

All groups in this section are abelian and written additively, namely $\mathbb{Z}$, $\mathbb{Z}_N$, and products of such groups. For them, the coset gH of Definition A.1 is written $g + H = \{g + h : h \in H\}$ instead. This is exactly the notation already used above, where the cosets of $H = 2\mathbb{Z}$ in $\mathbb{Z}$ were written $a + H$.

Definition A.2 (Constant and separating on cosets)**.** Let G be a group and S a set and H a subgroup of G. A function $f : G \to S$ is said to be *constant and separating on the cosets of H* if it takes the same value on every element of a given coset, and distinct values on distinct cosets. Formally,

$$f(g_1) = f(g_2) \iff g_1 H = g_2 H.$$

In other words, f perfectly labels the cosets of H: knowing $f(g)$ tells you exactly which coset g belongs to, but nothing more about g itself.

The *Hidden Subgroup Problem* (HSP) [49, 50] asks the following question. We are given a group and a function on it. The function is constant and separating on the cosets of some unknown hidden subgroup. The task is to determine that subgroup by querying the function. It is a generalization of well-known problems such as factoring or discrete logarithm. More precisely, the problem is stated as follows:

Definition A.3 (Hidden Subgroup Problem [51])**.** Let $f\colon G \to S$ be an efficiently computable function from a group G to a set S, where G is finite or finitely generated abelian. Suppose f is constant and separating on the cosets of some subgroup H. Determine the subgroup H.

The definition of [51] is stated for finite groups. We also allow finitely generated abelian groups such as $\mathbb{Z}$, which order finding requires.

We now instantiate Definition A.3 on the three computational problems that this paper studies: integer factoring, the finite-field discrete logarithm problem (FFDLP), and the elliptic-curve discrete logarithm problem (ECDLP). All three reduce to finding a hidden subgroup of an abelian group, and all three use a group exponentiation as the HSP oracle. They differ in three ways. First, the group order may or may not be known in advance. Second, the element order compares differently with the group order. Third, they need different numbers of exponent registers, or registers of different sizes.

Factoring.

Let $N = pq$ be an n-bit RSA integer with p, q unknown odd primes of similar size. The goal is to determine p and q.

Pick a random $g \in \mathbb{Z}_N^*$ and set $f(x) = g^x \bmod N$. Its period is the multiplicative order $r = \mathrm{ord}_{\mathbb{Z}_N^*}(g)$. The function f separates cosets of the hidden subgroup $H = \langle r \rangle = r\mathbb{Z} = \{0, \pm r, \pm 2r, \ldots\} \leq \mathbb{Z}$ (where $H \leq G$ means H is a subgroup of G):

$$\begin{aligned} f(x_1) = f(x_2) &\iff g^{x_1} \equiv g^{x_2} \pmod{N} \\ &\iff r \mid (x_1 - x_2) \\ &\iff x_1 + H = x_2 + H. \end{aligned} \tag{2}$$

Order finding is therefore the hidden subgroup problem for $(G, H) = (\mathbb{Z}, r\mathbb{Z})$. Recovering H is equivalent to recovering its generator r. In the modern Ekerå–Håstad variant, however, the algorithm no longer performs order finding. It reduces factoring to a short discrete logarithm in $\mathbb{Z}_N^*$, a group of unknown order [8, Sect. 5.2].

This HSP lives on the infinite group $\mathbb{Z}$, while the quantum circuit can only ever query f on a finite register. Lemma A.4 shows that the finite register preserves which inputs collide under f; its size controls an approximation error. The factoring algorithm is summarized as follows:

Given An n-bit RSA modulus $N = pq$ and a random $g \in \mathbb{Z}_N^*$. A $2n$-qubit exponent register, so that $2^{2n} \geq N^2$. A quantum black box that evaluates the function $f : |x\rangle\,|y\rangle \mapsto |x\rangle\,|(g^x \bmod N) \oplus y\rangle$.

Promise f is constant and separating on the cosets of $H = r\mathbb{Z} \leq \mathbb{Z}$, where $r = \mathrm{ord}_{\mathbb{Z}_N^*}(g)$ (Lemma A.4).

Output The order r, in time $O(n^3)$, using at most $O(n)$ qubits[3]. The factors p and q are then recovered classically via $\gcd(g^{r/2} \pm 1, N)$.

Finite register for factoring. The register size $2^{2n} \geq N^2 > r^2$ for the factoring case is chosen for purely computational reasons rather than structural ones. It guarantees $\approx 2^{2n}/r > r$ periods inside the register, so the resulting phase peaks are sharp. It also gives the approximation bound $1/(2 \cdot 2^{2n}) \leq 1/(2r^2)$ that continued-fraction recovery needs (Section A.3). Since r itself is unknown before the algorithm runs, the checkable condition $2^{2n} \geq N^2$ is used in its place.

Lemma A.4 (The finite register preserves coset collisions)**.** *Restrict f to the register $X = \{0, 1, \ldots, 2^{2n} - 1\}$. Two values $x_1, x_2 \in X$ satisfy $f(x_1) = f(x_2)$ if and only if $x_1 + H = x_2 + H$ in $\mathbb{Z}$. The collision classes $X \cap (a + H)$ are the cosets of a subgroup of $\mathbb{Z}_{2^{2n}}$ if and only if $r \mid 2^{2n}$.*

Proof. The first claim is Equation (2) applied to $x_1, x_2 \in X$. Restricting the domain removes pairs but changes no equivalence between the pairs that remain. For the second claim, write $2^{2n} = kr + s$ with $0 \leq s < r$. Since $r < N \leq 2^{2n}$, we have $k \geq 1$. The class of $a \in \{0, \ldots, r-1\}$ has $k+1$ elements if $a < s$ and k elements otherwise. The cosets of a subgroup all have the same size, so $s = 0$ is necessary. Conversely, if $r \mid 2^{2n}$, the classes are exactly the cosets of $r\mathbb{Z}_{2^{2n}}$.

The register therefore preserves which inputs collide, whatever its size. What its size controls is the Fourier side. When $r \nmid 2^{2n}$, the QFT over $\mathbb{Z}_{2^{2n}}$ concentrates the amplitude near the multiples $\mu\, 2^{2n}/r$ rather than on them. This is why the measured frequency only approximates μ/r, why the continued-fraction step of Section A.3 is needed, and why the register must satisfy $2^{2n} \gtrsim r^2$ (Figure 3).

Discrete logarithm.

Let p be a prime, g a generator of $\mathbb{Z}_p^*$, and $x = g^d \bmod p$ a given group element. The goal is to compute the discrete logarithm $d = \log_g x \in \mathbb{Z}_{p-1}$.

Let $G = \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$ and define

$$f : G \to \mathbb{Z}_p^*, \quad f(a,b) = g^a x^{-b} \bmod p = g^{a-bd} \bmod p.$$

The function f separates cosets of the hidden subgroup

$$\begin{aligned} H &= \left\{(a,b) \in \mathbb{Z}_{p-1}^2 : a \equiv bd \pmod{p-1}\right\} \\ &= \langle (d,1) \rangle \leq G. \end{aligned}$$

This holds because $f(a_1, b_1) = f(a_2, b_2)$ if and only if $a_1 - b_1 d \equiv a_2 - b_2 d \pmod{p-1}$, i.e. (a_1, b_1) and (a_2, b_2) lie in the same coset of H. Determining H is equivalent to determining the generator $(d, 1)$, which yields d directly. The overall process is summarized as follows:

Given The group $G = \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$ for a prime p, with generator $g \in \mathbb{Z}_p^*$. A target element $x = g^d \bmod p$. A quantum black box that evaluates the function $f : |a\rangle|b\rangle|y\rangle \to |a\rangle|b\rangle|g^a x^{-b} \bmod p \oplus y\rangle$.

Promise f is constant and separating on the cosets of $H = \langle (d,1) \rangle \leq G$, as shown above.

Output The discrete logarithm $d = \log_g x$, in time $O(n^3)$, and using at most $O(n)$ qubits[4].

In contrast to order finding, the discrete logarithm is already a hidden subgroup problem on a *finite* group, $G = \mathbb{Z}_{p-1}^2$. Its hidden subgroup $H = \langle (d,1) \rangle$ is a genuine subgroup of G, so no passage to $\mathbb{Z}$ is required (cf. Lemma A.4). Because the order $p-1$ is known, the exponent registers could in principle be matched to $\mathbb{Z}_{p-1}$. This requires a Fourier transform modulo $p-1$, which is not the power-of-two transform of standard circuits. Practical circuits instead use power-of-two registers of about the bit length of p, as Shor already did for general p [2, Sect. 7]. In factoring, by contrast, the unknown order r forces the over-provisioned register $2^{2n} \geq N^2$.

In the following, we distinguish two cases of the discrete logarithm problem, because the optimizations of Shor's algorithm developed separately for each. They are the finite field discrete logarithm problem (`FFDLP`) and the elliptic curve discrete logarithm problem (`ECDLP`). What distinguishes `ECDLP` from `FFDLP` is not the hidden subgroup but the cost of evaluating f, and we take it up in Section A.12.

A.2 The Noisy Hidden Subgroup Problem

Definition A.3 is an idealized promise: f is *exactly* constant on cosets of H and *exactly* distinct across distinct cosets. This is the right level of abstraction for stating what Shor's algorithm solves in principle, but it is not, in general, the object that a physical circuit actually queries. Realizing f on a finite or truncated register can perturb its values away from the exact promise (Section A.7). Any measurement-based read-out built on top of f can occasionally fail on a query (Sections A.5 and A.6). Both effects recur throughout the rest of this section. The finite register used for order finding preserves the coset collisions *exactly* (Lemma A.4). The perturbations therefore come from the successive optimizations of the oracle,

[3] The time-space complexity from the seminal Shor paper [2].

[4] Here again, the time-space complexity comes from [2].

namely smaller registers, compressed outputs and approximate arithmetic. Each introduces a controlled perturbation, either a bounded drift in the value of f or a read-out that can occasionally fail. Rather than handling each perturbation with a separate ad hoc argument, we capture the two ways the exact HSP promise can be relaxed in a single definition. Each later perturbation can then be located on the same two axes.

We introduce two parameters to make this relaxation precise. The first, that we denote ϵ, bounds how far a single query's output can drift from the value the exact promise would have given. It is a property of the oracle alone. The second, that we denote δ, bounds how often the algorithm's own read-out, whatever measurement and post-processing it performs on top of f, is allowed to fail. The two behave differently. ϵ does not depend on how the oracle is used, while δ depends entirely on the choice of read-out built on top of it. The same noisy oracle can be turned into a reliable or an unreliable instance of the HSP depending only on that choice. This device is the organizing lens that we introduce for comparing variants, not a standard notion. Related robustness analyses exist, for instance the approximate-periodicity arguments of [10] and [11, Sect. 2.2].

The per-shot failure rate δ splits into two kinds with very different costs, and we track them separately.

Definition A.5 (Read-out: abort and corruption)**.** A *read-out* R is the measurement, classical post-processing and acceptance test applied to a single invocation of an oracle $\mathcal{O}$ computing f. One shot returns a verdict $R \in \mathcal{Y} \cup \{\perp\}$, where $\perp$ is a *discard* symbol returned when the acceptance test fails. The verdict is random over the shot's measurement outcomes, together with any auxiliary randomness the read-out draws. The post-processing reads each $y \in \mathcal{Y}$ as a *constraint* on the hidden subgroup, that is, as a set C_y of subgroups of G compatible with y. The read-out must state C_y, including any tolerance it allows. A verdict y is *H-consistent* if $H \in C_y$, that is, if the constraint it is read as is true. Let $\mathcal{Y}_H = \{y \in \mathcal{Y} : H \in C_y\}$. A shot is a *success* if $R \in \mathcal{Y}_H$, an *abort* if $R = \perp$, and a *corruption* if $R \in \mathcal{Y} \setminus \mathcal{Y}_H$. The two failure probabilities and their sum,

$$\delta_\perp = \Pr[R = \perp], \qquad \delta_\times = \Pr[R \in \mathcal{Y} \setminus \mathcal{Y}_H],$$
$$\delta(R, \mathcal{O}) = \delta_\perp + \delta_\times = \Pr[R \notin \mathcal{Y}_H],$$

are taken as worst case over the instances of the problem, that is, over H and over the oracles that hide it. The subscripts name the verdict each rate tracks: $\perp$ for the discard symbol itself, and $\times$ for a wrong-but-accepted answer.

An abort is repaid by taking more shots. A corruption cannot be repaid that way, since the post-processing cannot tell it from a success, so it must instead be made rare or caught by a later check on a whole attempt (Lemma A.10). A success need not be informative. A true constraint can be weak, and the number of successes the post-processing needs is a separate quantity. The rates above are absolute. When a mechanism is added to a baseline read-out, we report the rates it adds.

Definition A.6 ((ϵ, δ)-noisy HSP)**.** Let G be a finite group, $H \le G$, and S a set carrying a metric d_S. Let $\mathcal{O}$ be an oracle computing $f\colon G \to S$, read by a read-out R (Definition A.5). For $\epsilon \ge 0$ and $\delta \in [0, 1)$, $\mathcal{O}$ is an (ϵ, δ)-*noisy instance of the HSP for H under R* if

(i) there is a function $f_0\colon G \to S$, constant and separating on the cosets of H in the sense of Definition A.2, with $d_S\big(f(g), f_0(g)\big) \le \epsilon/2$ for every $g \in G$;

(ii) $\delta(R, \mathcal{O}) \le \delta$.

The factor $\frac{1}{2}$ in (i) is a normalization. At $\epsilon = 0$ the oracle meets the exact promise of Definition A.3, and δ then measures the read-out alone. The metric d_S matters only when $\epsilon > 0$. For the residue-arithmetic oracle, $S = \mathbb{Z}_N$ and $d_S(u, v) = \min_{k \in \mathbb{Z}} |u - v - kN|/N$, the normalized modular distance of Equation (50).

Remark A.7 (When corruption needs value drift). An exact oracle ($\epsilon = 0$) can still corrupt. Whether it does depends on the read-out. Two situations rule it out. With an exact Fourier transform over G, every outcome lies in the subgroup $H^\perp$ orthogonal to H, so the constraint "$y \in K^\perp$" is always true of $K = H$. With an acceptance test that verifies the constraint, such as the check $g^{r'} \equiv 1$ in order finding, a false constraint becomes an abort instead. Outside these two situations an exact oracle can corrupt. The bad pairs of the Ekerå–Håstad read-out are an example (Section A.5). Value drift adds a second source of corruption. The same ϵ-drifting oracle turns its drift into corruptions ($\delta_\times \to 1$) under a full-precision measurement, yet the masked read-out of Section A.7 absorbs most of that drift into aborts. This is why δ is stated relative to R. Among the instances of Table 3, only the residue-arithmetic oracle has $\epsilon > 0$.

Remark A.8 (G need not contain a literal subgroup). Conditions (i) and (ii) of Definition A.6 only ever use the coset partition of G induced by H. Condition (i) uses it through the exact H-periodicity of f_0. Condition (ii) uses it through the H-consistent set $\mathcal{Y}_H$. That set, in turn, is fixed by the same partition. Neither needs a literal subgroup $H \le G$, so any G realizing the partition exactly will do, whether or not an actual subgroup sits inside it.

Four relaxations of the exact promise.

The remainder of this section develops the factoring case, starting with one baseline algorithm and three successive improvements. Section A.3 builds Shor's

full order-finding algorithm: a large exponent register, a full-width output register, and exact modular arithmetic. Three sections that follow each reduce one of these costs. Section A.5 shrinks the exponent register, so that a single measurement no longer determines the answer and it is recovered from repeated sampling instead. Section A.6 compresses the output register, hashing the oracle's output down toward a single qubit. Section A.7 makes the oracle's arithmetic space-efficient, computing the modular exponentiation approximately in a residue number system rather than exactly. None of the three reductions is free. Each relaxes the exact promise of Definition A.3 in a controlled way. It lets either the oracle's values drift or the read-out occasionally fail. These are the two parameters ϵ and δ of Definition A.6. Table 3 places the baseline and the three optimizations on them, and points to the section that derives each. The elliptic-curve problem of Section A.12 reuses these relaxations only in part and adds one the factoring problem never needed. Because it has no single best algorithm, its landscape is mapped separately there.

A.3 Shor's Algorithm

We first present the algorithm in its one-dimensional form, order finding for the factoring problem. This is the classical baseline and the clearest setting for the quantum core. The modern optimizations instead treat factoring in two dimensions, through a reduction to a short discrete logarithm that we give in Section A.4, immediately afterward.

Shor's algorithm is a three-stage algorithm, state preparation, oracle evaluation, and quantum Fourier transform, as depicted in Figure 4. It is surrounded by two classical components: pre-treatment and post-treatment. Its principle is the following: it performs superposition evaluations of the oracle function f to query it on all its possible inputs, namely all the exponentiations $f\colon e \to g^e \bmod N$ for some randomly selected g. It then finds the smallest period of f, namely its order, using the quantum Fourier transform and some classical post-processing.

State preparation.

The first stage of the algorithm is to prepare two quantum registers, ψ_1 and ψ_2. Register ψ_1 holds the superposition of all 2^{n_e} possible exponents, where n_e is the width of the exponent register. Register ψ_2 holds the accumulator used during the computation of the oracle O_f, namely an n-qubit register initialized at 1, for $n = \log N$. For order finding, ψ_1 is the $2n$-qubit register of Lemma A.4, so $n_e = 2n = O(\log N)$. After state preparation, ψ_1 and ψ_2 hold the following values:

$$\psi_1\colon \left|0^{n_e}\right\rangle \xrightarrow{H^{\otimes n_e}} \frac{1}{\sqrt{2^{n_e}}} \sum_{x=0}^{2^{n_e}-1} |x\rangle \tag{3}$$

$$\psi_2\colon \left|0^{n-1}1\right\rangle \xrightarrow{I^{\otimes n}} \left|0^{n-1}1\right\rangle \tag{4}$$

As shown in Figure 4, the two registers are easily and efficiently initialized. ψ_1 is put into uniform superposition using a Hadamard layer, and ψ_2 is set to the basis state $\left|0^{n-1}1\right\rangle$.

Oracle evaluation.

The second stage evaluates the modular exponentiation $g^e \bmod N$ and accumulates the result into the second register ψ_2. The exponent e is encoded in the first register ψ_1, and the base g is a fixed classical constant. The high-level process is depicted in Figure 5. It is structurally identical to the controlled stage of quantum phase estimation of a unitary U, which maps $|j\rangle\,|u\rangle \mapsto |j\rangle\,U^j\,|u\rangle$ (see Section 5.2 of the Nielsen and Chuang [38]). Writing the exponent in binary, $e = \sum_i 2^i e_i$, with e_0 the least significant bit, the exponentiation can be rewritten as

$$g^e \bmod N = \prod_i \left(g^{2^i}\right)^{e_i} \bmod N, \tag{5}$$

so the oracle reduces to a sequence of modular multiplications by the constants $g^{2^i} \bmod N$, each controlled by the corresponding exponent qubit e_i. Throughout this paper, integers are written in this little-endian form, and circuits draw bit 0 on the top wire.

In a generic phase-estimation setting, the controlled powers U^{2^i} are obtained by repeated application of U, so applying U^{2^i} costs on the order of 2^i applications of U. The total cost is then dominated by the deepest power, and grows geometrically. Shor's algorithm avoids this blow-up because the relevant operator is multiplication in $(\mathbb{Z}/N\mathbb{Z})^*$. Each constant $g^{2^i} \bmod N$ is precomputed on a classical computer, and the corresponding controlled multiplication has the same cost regardless of i. Specifically, multiplying an n-bit register by a fixed n-bit constant modulo N has a cost that does not depend on the value of that constant. Consequently an n_e-qubit exponent register, spanning 2^{n_e} values, is handled by only n_e controlled multiplications of uniform cost, rather than by explicitly forming U^{2^i} through repeated squaring. This is what makes the modular exponentiation and hence the whole phase-estimation core of Shor's algorithm efficient.

Quantum Fourier transform.

The third stage applies the inverse quantum Fourier transform to the exponent register ψ_1 and measures it, extracting the period r from the periodicity created

Instance	ϵ	$\delta = \delta_\perp + \delta_\times$		Read-out R	In Section A.9
		$\delta_\perp$ *abort*	$\delta_\times$ *corrupt*		
Finite register (order finding, baseline)	0	$\leq 1 - \frac{4}{\pi^2}\frac{\varphi(r)}{r}$	0	measure the exponent register, continued fractions, accept r' iff $g^{r'} \equiv 1$	no. Superseded by the short-DLP reduction and the register of Section A.5
Ekerå–Håstad good pair (Section A.5)	0	0	$\leq 7/8$	measure (j, k), accept every pair. A bad pair cannot be detected per shot, so it corrupts	register shape only. Read-out replaced by graded post-processing [52]
May–Schlieper hash (Section A.6)	0	$\leq 2^{-t}$	0	draw $h \sim \mathcal{H}_t$, measure $h \circ f$, group by equality	Indirectly: masking is used instead
RNS $\tilde{V}$ + masking (Section A.7)	$O(\lvert P\rvert \, \ell \, 2^{-\kappa})$	$\approx S$	$\approx \epsilon/S$	draw mask, add $\tilde{V}$, measure (mask width SM)	Yes

Table 3: The four (ϵ, δ)-noisy HSP relaxations studied in this section, with the per-shot failure resolved into aborts $\delta_\perp$ (a wasted shot) and corruptions $\delta_\times$ (a wrong constraint accepted). Each row lists the failure that mechanism introduces when taken on its own. The rows are not stages of one pipeline, and some of them replace one another. The last column states which of them the RSA-2048 pipeline of Section A.9 uses. Corruption occurs in two rows. In the Ekerå–Håstad row an exact oracle corrupts through bad pairs, which no per-shot test can detect. In the RNS row the oracle values themselves drift ($\epsilon > 0$). The other two rows can fail only by aborting (Remark A.7). The RNS optimum $S = \sqrt{\epsilon}$ is the point that balances $\delta_\perp$ against $\delta_\times$, giving $\delta \approx 2\sqrt{\epsilon}$. Both RNS rates are heuristic estimates (Section A.7).

by the oracle. To understand why this works, think of the oracle not as a black box computing $g^e \bmod N$. It is better seen as the repeated application of a unitary operator, whose spectral structure encodes r directly.

i The eigenvalue decomposition. Define the operator $U_g : |x\rangle \mapsto |gx \bmod N\rangle$ acting on the output register ψ_2. Its action on the output register is multiplication by g modulo N. Since the order of g is r, applying U_g exactly r times returns every state to itself:

$$U_g^r \, |x\rangle = |g^r x \bmod N\rangle = |x\rangle \,, \quad \text{so} \quad U_g^r = I.^5 \tag{6}$$

The polynomial $X^r - 1$ therefore annihilates U_g. So the eigenvalues of U_g are among the roots of $X^r - 1$, namely the r-th roots of unity. Writing $\omega = e^{2\pi i/r}$, the eigenvalues λ and their eigenvectors v are:

$$\begin{aligned} &U_g \, |v_\mu\rangle = \lambda_\mu \, |v_\mu\rangle \,, \quad \lambda_\mu = \omega^\mu, \\ &|v_\mu\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \omega^{-\mu k} \left|g^k \bmod N\right\rangle, \mu \in \{0, \ldots, r-1\}, \end{aligned} \tag{7}$$

as illustrated in Figure 6. These r vectors diagonalize U_g on the r-dimensional subspace spanned by the orbit $\{\left|g^k \bmod N\right\rangle : 0 \leq k < r\}$ of $|1\rangle$. Being among the roots of $X^r - 1$ bounds how many distinct eigenvalues U_g can have. Since $U_g^r = I$ on the whole N-dimensional space (footnote above), U_g is diagonalizable everywhere. Its remaining $N - r$ eigenvectors simply reuse these same r values with higher multiplicity, rather than requiring new ones. U_g, being a permutation of $\{0, \ldots, N-1\}$, maps every orbit strictly into itself. Starting at $|1\rangle$ confines the state, for the whole computation, to the one r-dimensional orbit it starts in, and the eigenvectors of every other orbit never acquire any amplitude. The phase of the μ-th eigenvalue is μ/r: it encodes r directly. The task of period finding therefore reduces to estimating that phase. Preparing any specific eigenvector $|v_\mu\rangle$ would require knowing r, the very quantity we are trying to find. The key observation that breaks this circularity is the identity

$$\begin{aligned} \frac{1}{\sqrt{r}} \sum_{\mu=0}^{r-1} |v_\mu\rangle &= \frac{1}{r} \sum_{\mu=0}^{r-1} \sum_{k=0}^{r-1} \omega^{-\mu k} \left|g^k \bmod N\right\rangle \\ &= \frac{1}{r} \sum_{k=0}^{r-1} \Big(\sum_{\mu=0}^{r-1} \omega^{-\mu k} \Big) \left|g^k \bmod N\right\rangle = |1\rangle \,. \end{aligned} \tag{8}$$

The first equality inserts the definition of $|v_\mu\rangle$. The second groups the terms by basis state. The inner sum adds up the r-th roots of unity $\omega^{-k\mu}$. It equals r for $k = 0$ and vanishes for every other k, so all basis states cancel except $\left|g^0 \bmod N\right\rangle = |1\rangle$. The state $|1\rangle$, preparable by setting the output register to the integer 1, is already a uniform superposition over these r eigenvectors. By linearity, running phase estimation on $|1\rangle$ runs it simultaneously on all eigenvectors at once, without ever having to construct any individual

[5] Note that this holds on the full computational basis $\{0, \ldots, N-1\}$, not only on $\mathbb{Z}_N^*$. Since $g^r \equiv 1 \pmod N$, we have $N \mid x(g^r - 1)$ for every integer x. Hence $g^r x \equiv x \pmod N$, regardless of whether $\gcd(x, N) = 1$.

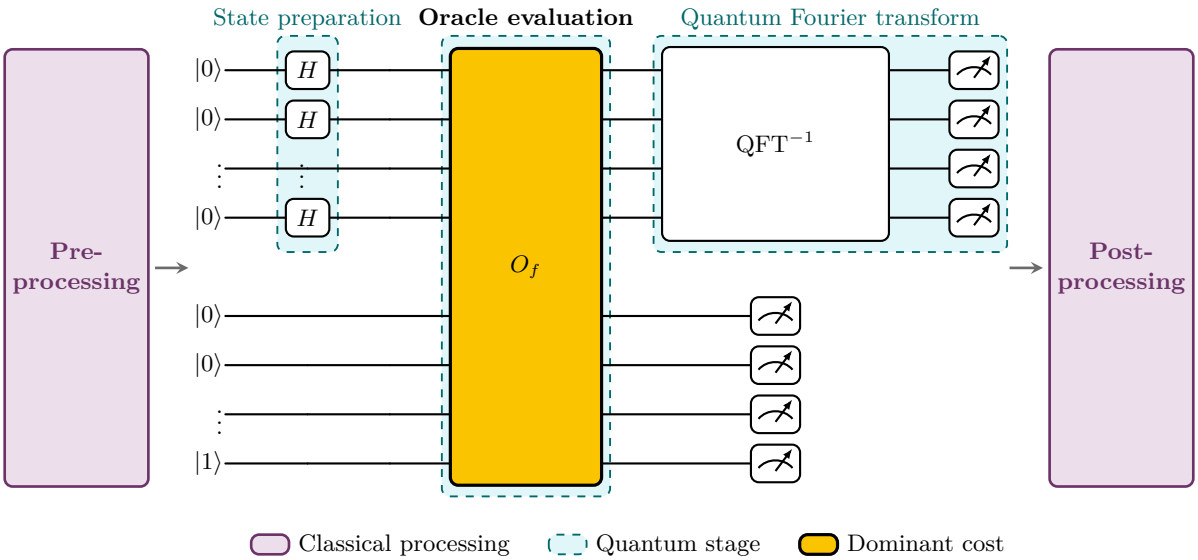


Figure 4: Textbook Shor's algorithm, end to end. The exponent register ψ_1 (top, $n_e = 2n$ qubits) is put in uniform superposition by Hadamard gates. The accumulator ψ_2 (bottom, n qubits) starts in the integer 1, that is $\left|0^{n-1}1\right\rangle$. The oracle O_f maps $|e\rangle\,|1\rangle$ to $|e\rangle\,|g^e \bmod N\rangle$. The inverse quantum Fourier transform is then applied to ψ_1, which is measured. Measuring ψ_2 is optional. By the principle of deferred measurement it does not change the outcome distribution of ψ_1, and it is drawn here because it simplifies the analysis. Classical pre- and post-processing choose g and recover r from the measured value.

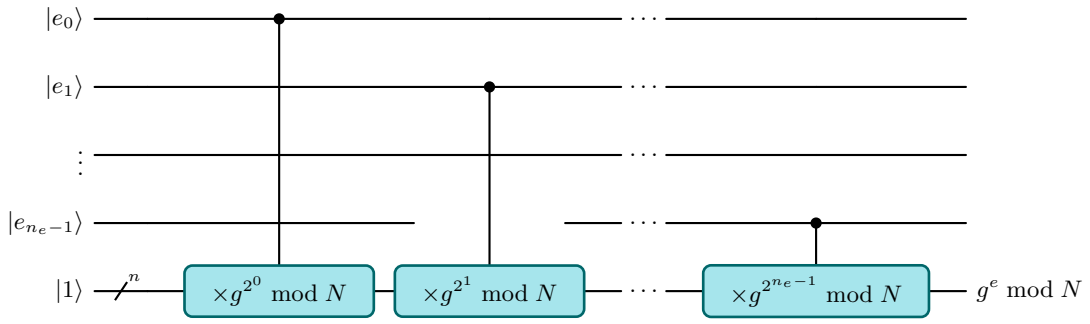


Figure 5: Quantum circuit of the oracle function evaluation. Each of the controlled blocks has the same computational cost, showcasing one of the advantages of Shor's algorithm: each constant has equal cost since the constant values $g^{2^i} \bmod N$ are precomputed classically.

$|v_\mu\rangle$.

ii Running phase estimation on all eigenvectors at once. The oracle applies controlled powers of U_g: since $|g^e \bmod N\rangle = U_g^e\,|1\rangle$, it acts as $|e\rangle\,|1\rangle \mapsto |e\rangle\,U_g^e\,|1\rangle$. Feeding in $|1\rangle = \frac{1}{\sqrt{r}}\sum_\mu |v_\mu\rangle$ and using $U_g^e\,|v_\mu\rangle = \omega^{\mu e}\,|v_\mu\rangle$,

$$\begin{aligned}\frac{1}{\sqrt{2^{n_e}}}\sum_{e=0}^{2^{n_e}-1}|e\rangle\,|g^e \bmod N\rangle &= \frac{1}{\sqrt{2^{n_e}}}\sum_{e=0}^{2^{n_e}-1}|e\rangle\,U_g^e\,|1\rangle \\ &= \frac{1}{\sqrt{r}}\sum_{\mu=0}^{r-1}|\phi_\mu\rangle\,|v_\mu\rangle\,,\end{aligned} \tag{9}$$

with $|\phi_\mu\rangle := \frac{1}{\sqrt{2^{n_e}}}\sum_{e=0}^{2^{n_e}-1} e^{2\pi i\mu e/r}\,|e\rangle$. In each branch μ, this single entangled state correlates the eigenvector $|v_\mu\rangle$ in ψ_2 with a state $|\phi_\mu\rangle$ in ψ_1. The amplitude of $|\phi_\mu\rangle$ has the same magnitude on every $|e\rangle$. Its phase $2\pi\mu e/r$ advances by a fixed step each time e increases by one, a *phase ramp* of frequency μ/r. The equal weights $1/\sqrt{r}$ are inherited directly from $|1\rangle = \frac{1}{\sqrt{r}}\sum_\mu |v_\mu\rangle$: every eigenphase μ/r is present with the same amplitude. This is the concrete meaning of "running phase estimation on $|1\rangle$ probes all eigenvectors at once": one run of the circuit realizes all r phase-estimation instances $|\phi_\mu\rangle$ in superposition.

By the principle of deferred measurement (Nielsen and Chuang [38], §4.4), one may measure ψ_2 first,

the implicit measurement of Figure 4. Obtaining a value g^{e_0} leaves ψ_2 in the computational-basis state $|g^{e_0}\rangle = \frac{1}{\sqrt{r}} \sum_{\mu=0}^{r-1} \omega^{\mu e_0} |v_\mu\rangle$, an equal-weight combination of *all* r eigenvectors, not a single one. Correspondingly, ψ_1 collapses to the periodic superposition

$$|\psi_1\rangle \longrightarrow \frac{1}{\sqrt{A}} \sum_{k=0}^{A-1} |e_0 + kr\rangle, \qquad A = \left\lceil \tfrac{2^{n_e} - e_0}{r} \right\rceil \approx \frac{2^{n_e}}{r},$$

over the $A \approx 2^{n_e}/r$ exponents consistent with it. Measuring ψ_2 therefore does not single out one eigenvector. The inverse QFT resolves this periodic superposition of A equally spaced spikes into the same r frequency peaks as the equal-weight eigenvector sum above. We follow the eigenvector form in what follows.

iii Phase estimation read-out. The inverse QFT of size 2^{n_e} resolves each phase ramp $|\phi_\mu\rangle$ into a sharp frequency peak. With the phase-estimation convention $\mathrm{QFT}^{-1}_{2^{n_e}} |x\rangle = \frac{1}{\sqrt{2^{n_e}}} \sum_c e^{-2\pi i x c / 2^{n_e}} |c\rangle$,[6]

$$\mathrm{QFT}^{-1}_{2^{n_e}} |\phi_\mu\rangle = \frac{1}{2^{n_e}} \sum_{c=0}^{2^{n_e}-1} \Big(\sum_{e=0}^{2^{n_e}-1} e^{2\pi i e\,(\mu/r - c/2^{n_e})} \Big) |c\rangle,$$

a finite geometric series in e. When $\mu/r - c/2^{n_e}$ is close to an integer, its terms add constructively and the sum reaches magnitude near 2^{n_e}, so the coefficient of $|c\rangle$ approaches 1 in modulus. Elsewhere the terms largely cancel, which produces a sharp peak at $c \approx \frac{\mu\, 2^{n_e}}{r}$.[7] Because the QFT acts on ψ_1 alone, it transforms every branch at once,

$$\frac{1}{\sqrt{r}} \sum_{\mu=0}^{r-1} |\phi_\mu\rangle\, |v_\mu\rangle \xrightarrow{\mathrm{QFT}^{-1} \otimes I} \frac{1}{\sqrt{r}} \sum_{\mu=0}^{r-1} \big(\mathrm{QFT}^{-1}_{2^{n_e}} |\phi_\mu\rangle\big)\, |v_\mu\rangle, \tag{10}$$

producing r peaks, one per eigenphase, each still weighted $1/r$. Measuring ψ_1 (the explicit measurement of Figure 4) therefore returns an integer c with

$$\frac{c}{2^{n_e}} \approx \frac{\mu}{r}, \qquad \mu \in \{0, \ldots, r-1\} \text{ uniform.} \tag{11}$$

Since c is the nearest integer to $\frac{\mu\, 2^{n_e}}{r}$, the error satisfies $|\frac{c}{2^{n_e}} - \frac{\mu}{r}| \le 2^{-n_e - 1}$. The standard phase-estimation accuracy bound (Nielsen and Chuang [38], §5.2) gives each peak probability at least $4/\pi^2 \approx 0.405$, so with constant probability the measured c is a best approximation to μ/r. The uniformity of μ is a property of the spectrum, not of any measurement randomness: the r peaks carry equal weight precisely because $|1\rangle = \frac{1}{\sqrt{r}} \sum_\mu |v_\mu\rangle$ weights the r eigenvectors equally.

iv Semi-classical implementation. Because each qubit of ψ_1 is measured immediately after its Hadamard gate, the QFT can be implemented in its semi-classical form [53]. The controlled phase rotations $R_k^\dagger |b\rangle = e^{-2\pi i b / 2^k} |b\rangle$, for $b \in \{0, 1\}$, which the inverse transform requires on the not-yet-measured qubits, are replaced by rotations *classically* controlled on the already-measured outcomes. This fixes the order of the measurements. The qubit e_{n_e-1}, which controlled the highest power $g^{2^{n_e-1}}$, is measured first, and its outcome is the least significant bit c_0 of c. The others follow in decreasing order of the power they controlled [53], so that e_{n_e-1-i} yields c_i. This order is compatible with the oracle. The controlled multiplications of Figure 5 commute, since multiplication modulo N is commutative, so they can be applied in the order in which the semi-classical transform measures their control qubits. Successive classically controlled rotations on the same qubit merge into a single rotation, leaving one Hadamard, one phase gate, and one measurement per qubit. This defers to the classical world everything that need not be quantum, exactly as the oracle deferred the powers $g^{2^i} \bmod N$ to classical precomputation. It reduces the QFT to a cost linear in n_e, negligible next to the exponentiation. The resulting circuit is shown in Figure 7.

v Recycling the exponent register. The semi-classical form also saves space. Each qubit of ψ_1 is needed only to control its own multiplications. It is then rotated, measured, and never used again. Since the controlled multiplications commute, they can be performed in the order in which the semi-classical transform measures the qubits. The three stages can therefore be interleaved. One prepares a single control qubit with a Hadamard, applies the controlled multiplication it steers, applies the classically controlled rotation and the final Hadamard, measures the qubit, and resets it for the next exponent bit [54–56]. With windowing, a block of w_e control qubits is recycled in the same way. The exponent register then never exists in full. Only one window of it is alive at any time, whatever the value of n_e. The cost analysis below keeps the three stages separate for clarity, and states the qubit count both with and without recycling.

The classical recovery of r from $c/2^{n_e}$ follows.

Classical post-processing

The classical step that follows the measurement depends on the modulus of the quantum Fourier transform. When the order of the group is known, the

[6] The standard phase-estimation convention (Nielsen and Chuang [38], §5.2), under which a frequency-μ/r ramp is resolved to a peak at $c \approx \mu\, 2^{n_e}/r$. It is the conjugate of the opposite sign choice and has no direct effect on the recovery of r.

[7] Explicitly, the sum evaluates to the Dirichlet kernel $\big|\sin\big(\pi 2^{n_e}(\mu/r - c/2^{n_e})\big) \big/ \sin\big(\pi(\mu/r - c/2^{n_e})\big)\big| / 2^{n_e}$, but only its peak location and sharpness are used below.

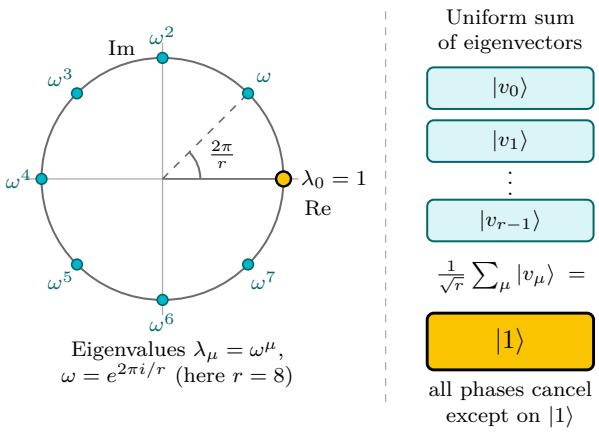


Figure 6: Left: the eigenvalues $\lambda_\mu = \omega^\mu$ of U_g, the r-th roots of unity (here $r = 8$). Right: the corresponding eigenvectors $|v_\mu\rangle$ of Equation (7) sum to $|1\rangle$, $\frac{1}{\sqrt{r}} \sum_\mu |v_\mu\rangle = |1\rangle$ (Equation (8)). This is an exact identity. In the sum, the phases cancel on every basis state except $|1\rangle$.

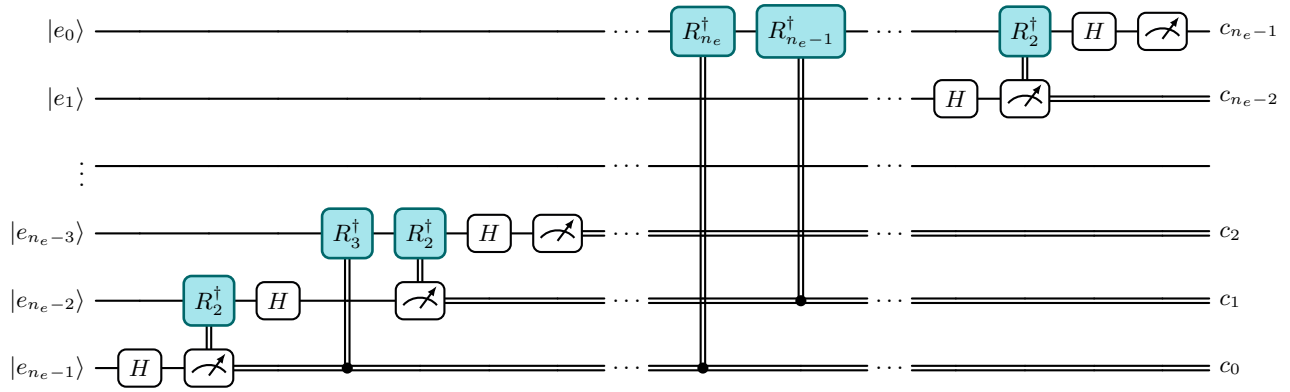


Figure 7: Semi-classical inverse QFT on the exponent register ψ_1. The qubits are measured one at a time, from the bottom up. The qubit e_{n_e-1}, which controlled the highest power $g^{2^{n_e-1}}$, is measured first and gives the least significant bit c_0 of the outcome $c = \sum_i 2^i c_i$. In general, e_{n_e-1-i} gives c_i. Each later qubit is first corrected by rotations $R^\dagger_k$, classically controlled on the outcomes already measured, and then measured. The rotations on one qubit are drawn separately for clarity. In practice they merge into a single rotation, so each qubit needs one Hadamard, one rotation and one measurement.

transform can in principle be taken modulo that order, and it is then exact. This needs a quantum Fourier transform for an arbitrary modulus [57], which is more involved than the power-of-two transform and is not what implementations use. In practice the transform is taken modulo a power of two, whether the order is known or not, and it only approximates the ideal one. The register must then be large compared with the order. We describe the exact case first, because it shows the structure of the outcome most clearly.

vi Known order: exact transform (idealization). In the discrete-logarithm problem, the order $p - 1$ of $\mathbb{Z}_p^*$ is known. Assume an exact quantum Fourier transform over $G = \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1}$ is available [57]. The measurement then yields a pair (j, k) uniformly distributed over the subgroup orthogonal to $H = \langle (d, 1) \rangle$, that is, over the solutions of

$$dj + k \equiv 0 \pmod{p-1}, \tag{12}$$

and every other outcome has amplitude zero. Whenever j is invertible modulo $p - 1$, the discrete logarithm follows directly as $d = -k\, j^{-1} \bmod (p-1)$, with no approximation. Since j is uniform over $\mathbb{Z}_{p-1}$, this happens with probability $\varphi(p-1)/(p-1) = \Omega(1/\log\log p)$ [58]. Shor treats this exact case only when $p - 1$ is smooth, so that the transform modulo $p - 1$ is easy to build [2, Sect. 6]. In that case the Pohlig–Hellman algorithm already solves the problem classically. For general p he uses a power-of-two register q with $p \le q < 2p$ [2, Sect. 7]. The outcome then only lies close to the line $dj + k \equiv 0 \pmod{p-1}$, and recovering d needs an approximate analysis of the same kind as for order finding below. Every implementation we cite follows this power-of-two route, including those of Sections A.5 and A.12.

vii Unknown order: transform modulo a power of two. In order finding, the order r is precisely the unknown, so the transform cannot be

matched to it. It is instead taken over $\mathbb{Z}_q$ with $q = 2^{n_e}$. Since r does not in general divide q, the off-peak amplitudes no longer cancel exactly. The outcome c concentrates near the multiples $q\mu/r$, with some spread. Choosing $q \geq N^2 > r^2$, phase estimation gives, with probability at least $4/\pi^2$,

$$\left|\frac{c}{q} - \frac{\mu}{r}\right| \leq \frac{1}{2q} < \frac{1}{2r^2}. \tag{13}$$

The measurement reveals c and q, but neither μ nor r individually. They are recovered with a classical result on continued fractions due to Legendre [59]. If a real number x satisfies $|x - \mu/r| < 1/(2r^2)$, then μ/r, in lowest terms, is a convergent of the continued fraction expansion of x. Moreover, two distinct fractions with denominators below N differ by more than $1/N^2 \geq 1/q$. Thus μ/r is the only such fraction within $1/(2q)$ of c/q, and it is the last convergent of c/q whose denominator is below N.

It therefore suffices to expand c/q as a continued fraction,

$$\frac{c}{q} = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}. \tag{14}$$

One then computes its successive convergents. This yields μ/r in lowest terms, whose denominator is the order r provided $\gcd(\mu, r) = 1$. By Mertens' third theorem [58], the proportion of integers in $\{0, \ldots, r-1\}$ coprime to r satisfies $\varphi(r)/r = \Omega(1/\log\log r)$. A uniformly random μ is therefore coprime to r with probability $\Omega(1/\log\log r)$, and $O(\log\log r)$ repetitions suffice in expectation.

Efficient implementation of the oracle.

The modular exponentiation $|e\rangle|1\rangle \mapsto |e\rangle|g^e \bmod N\rangle$ is built up through a hierarchy of nested decompositions and improved representations. Exponentiation is reduced to multiplications (as seen in Figure 5), and multiplications to product-additions. Modular additions use the coset representation [60, 61], and the arithmetic is windowed for efficiency [62, 63]. The overall implementation is detailed at great length in [3] and is depicted in Figure 8. We give hereafter a brief overview of the full computation of the modular exponentiation using these different techniques.

viii From exponentiation to multiplications. Writing e in binary form $e_{n_e-1} \ldots e_1 e_0$, the exponentiation factors as a product of controlled multiplications,

$$g^e = \prod_{i=0}^{n_e-1} \left[g^{2^i}\right]^{e_i}, \tag{15}$$

each factor $\left[g^{2^i}\right]^{e_i}$ being applied to the accumulator register conditioned on the exponent bit e_i. This yields n_e modular multiplications.

ix From multiplications to product-additions. Each multiplication $x \mapsto x \cdot g^{2^i e_i} \bmod N$ is realized with two controlled product-additions and one controlled swap, following the out-of-place scheme $|y\rangle|z\rangle \mapsto |y\rangle|z + y\gamma \bmod N\rangle$. Expanding the product over the bits y_j of y,

$$y \cdot \gamma = \sum_{j=0}^{n-1} \left[\gamma 2^j\right] y_j, \tag{16}$$

turns each product-addition into a sequence of n controlled modular additions. Concretely, the procedure has three steps. First, the product-addition writes $\bar{x} = x\, g^{2^i}$ into an ancilla register. Second, it subtracts $\bar{x}\, g^{-2^i} = x$ back out of the input register, leaving it at 0. Third, a controlled swap exchanges the two registers, so that the product $x\, g^{2^i e_i}$ ends up in the accumulator with the ancilla reset to $|0\rangle$.

x Coset representation for the modular additions. Rather than implementing each modular addition with the usual add, compare, correct, uncompute sequence, the target register is stored in the *coset representation* [60, 61]. The basic idea is to extend the n-qubit register holding z with l additional qubits and to encode the integer z into the uniform superposition over 2^l coset shifts,

$$|z\rangle_{\text{coset}} = \frac{1}{\sqrt{2^l}} \sum_{k=0}^{2^l-1} |z + kN\rangle\,. \tag{17}$$

The shift by N maps the superposition to itself except for the two end terms ($k = 0$ lost, $k = 2^l$ gained). This invariance implies that adding γ modulo N is well approximated by a plain addition modulo 2^{n+l},

$$\begin{aligned} &\frac{1}{\sqrt{2^l}} \sum_{k=0}^{2^l-1} \left|(z + \gamma + kN) \bmod 2^{n+l}\right\rangle \\ \approx\ &\frac{1}{\sqrt{2^l}} \sum_{k=0}^{2^l-1} |(z + \gamma \bmod N) + kN\rangle\,, \end{aligned} \tag{18}$$

where the approximation fails only at the boundary of the superposition. Write $z' = (z + \gamma) \bmod N$, so that $z + \gamma$ equals either z' or $z' + N$. The plain addition then maps the 2^l shifts $k \in \{0, \ldots, 2^l - 1\}$ of z either to the same shifts of z' or to the shifts $k + 1$. An addition therefore never scrambles the encoding. It only slides the window of shifts by at most one position. After n_{add} additions the window has slid by at most n_{add} positions. It then shares at least $2^l - n_{\text{add}}$ of its 2^l terms with the ideal coset state, so the overlap of the two states is at least $1 - n_{\text{add}}\, 2^{-l}$. The trace distance between the actual and the ideal state is then at most $\sqrt{2\, n_{\text{add}}\, 2^{-l}}$. The exponentiation performs $n_{\text{add}} = O(n_e n)$ additions, so keeping this distance below a fixed constant requires

$l = \log_2 n_{\text{add}} + O(1) = \Theta(\log n)$. The error falls exponentially in l, which is why a logarithmic padding suffices. A rigorous bound is given in [61], and its application to the full exponentiation, including the uncomputation within each multiplication, in [25].

xi Windowed arithmetic. Finally, the exponent bits are grouped into windows of w_e bits and the multiplicand bits into windows of w_l bits [63]. Each window value indexes a classically precomputed constant $T_k = 2^j k_2\, g^{2^i k_1} \bmod N$, with $k = (k_1, k_2)$, $k_1 = e_{i:i+w_e}$ and $k_2 = x_{j:j+w_l}$. Here i indexes the exponent window and j the multiplicand window, and $y_{a:b} = \lfloor y/2^a \rfloor \bmod 2^{b-a}$ denotes the integer formed by bits a to $b-1$ of y. It is loaded into an ancilla via a quantum table lookup $|k\rangle|0\rangle \mapsto |k\rangle|T_k\rangle$, added unconditionally to the accumulator, and then unloaded. This divides the number of multiplications by w_e and the number of additions per multiplication by w_l, reducing the leading cost of the exponentiation from $O(n_e n^2)$ to $O(n_e(n+l)^2/(w_e w_l))$ Toffoli gates [3][8].

The implementation of the exponentiation oracle illustrates one of the key considerations when realizing complex arithmetic operations on a quantum computer. The goal is to defer as much of the computation as possible to the classical world, reducing the time-space cost of the quantum part.

Overall cost.

We now summarize the space-time requirements of the three stages, in terms of logical qubits and Toffoli gates, as functions of the modulus size $n = \log N$ and the exponent size n_e. Please note that this cost is not linked to up-to-date results on Shor where algorithmic optimizations have been made but to the seminal paper, where we only augmented it with arithmetic tricks. The aim is to understand how and where the different improvements of the modern Shor algorithm make their impact.

xii State preparation. The first stage acts with Hadamard gates only: a layer of n_e Hadamards on ψ_1 and none on ψ_2. It uses no Toffoli gates and requires the two registers to be allocated, namely the n_e-qubit exponent register and the $(n+l)$-qubit accumulator. With recycling, only w_e qubits of the exponent register are allocated at a time, and their Hadamards are applied just before the window is used. Its cost is therefore negligible, contributing $O(n_e)$ Clifford gates and no non-Clifford cost.

xiii Oracle evaluation. The second stage dominates the entire algorithm. The exponentiation is a chain of n_e controlled modular multiplications. Each multiplication is two product-additions over $(n+l)$-qubit registers, and each product-addition is a sequence of modular additions. Windowing the exponent into blocks of w_e bits reduces the number of multiplications by a factor w_e. Windowing the multiplicand into blocks of w_l bits reduces the number of additions per multiplication by a factor w_l. This leaves a leading Toffoli cost of

$$T_{\text{oracle}} = O\left(\frac{n_e\,(n+l)^2}{w_e\,w_l}\right), \tag{19}$$

as derived in [3]. The space requirement is set by the two registers and the lookup ancilla. The input register $|x\rangle_{\text{coset}}$ and the accumulator each use $n+l$ qubits. The table lookup for a window of $w_e + w_l$ address bits temporarily allocates an n-qubit ancilla to hold T_k. The stage therefore uses

$$Q_{\text{oracle}} = 2(n+l) + O(n) \tag{20}$$

logical qubits. The coset padding is chosen as $l = O(\log n)$, which keeps the accumulated error of all $O(n_e\, n)$ additions below a fixed target.

xiv Quantum Fourier transform. The third stage acts only on the n_e-qubit exponent register. In its semi-classical form each qubit contributes one Hadamard, one classically controlled phase rotation, and one measurement, so the transform costs $O(n_e)$ single-qubit rotations, negligible but non-Clifford, and allocates no qubits beyond the exponent register already present. Its cost is thus negligible next to the oracle.

xv Total. Non-Clifford cost is incurred almost entirely by the oracle, so the full algorithm requires

$$T_{\text{total}} = O\left(\frac{n_e\,(n+l)^2}{w_e\,w_l}\right) \quad \text{Toffoli gates,} \tag{21}$$

and, when the exponent register is held in full,

$$Q_{\text{total}} = n_e + 2(n+l) + O(n) = O(n) \quad \text{logical qubits,} \tag{22}$$

using $l = O(\log n)$. With recycling, the n_e term drops to w_e, the size of one window, so $Q_{\text{total}} = w_e + 2(n+l) + O(n)$ whatever the value of n_e. In the plain (unwindowed) regime $w_e = w_l = 1$ and with $l = O(\log n)$, the Toffoli count reaches $O(n_e\, n^2)$. For order finding the exponent is taken as $n_e = 2n$, recovering the familiar $O(n^3)$ scaling of Shor's algorithm.

We now have a clear view of Shor's algorithm and how it solves the HSP for factoring. The modern, state-of-the-art algorithm lowers its cost significantly by composing three results: the input-register reduction of Ekerå–Håstad [8], the output-register reduction of May–Schlieper [9], and the oracle compression of Chevignard, Fouque and Schrottenloher [10]. All three operate in two dimensions. The first step is the reduction that puts factoring there, to which we now turn.

[8] This is taking the assumption that the lookup table can be implemented at a similar cost as an addition, which is true with proper windowing sizes and fan-out of cost $O(1)$.

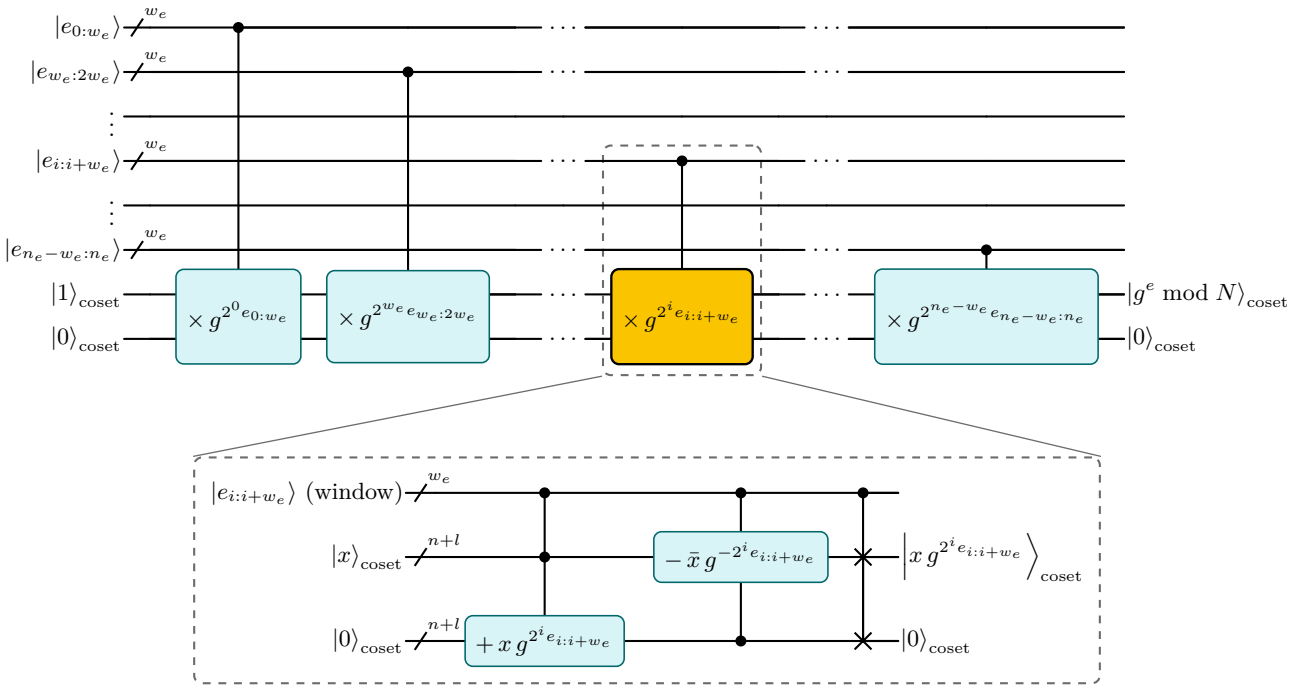


Figure 8: Hierarchical decomposition of the modular exponentiation. Top: the exponentiation $g^e \bmod N$ is a chain of $\lceil n_e/w_e \rceil$ controlled modular multiplications on the coset-represented accumulator. Bottom: one controlled multiplication, realized out-of-place as described in the text above. Each product-addition is itself expanded into windowed modular additions $+T_k \bmod N$ via table lookups, as in [3]. Each wire marked w_e is a window of w_e exponent qubits. A dot on a window means that its value selects the multiplier applied, not a control on a single qubit. All arithmetic is modulo N, and the $\bmod N$ is omitted inside the boxes.

A.4 From Factoring to Short Discrete Logarithm Problem

The order-finding algorithm of the previous section works in one dimension. The optimizations that follow all operate in two dimensions, and they reach factoring only through the reduction of factoring to a short discrete logarithm developed here. For factoring, we rely on the Ekerå–Håstad period-finding algorithm [8], a recent variant of Shor's algorithm. The core observation of their paper is the following. The multiplicative group of integers modulo N, $\mathbb{Z}_N^*$, has order $\phi(N) = (p-1)(q-1)$. By definition of the Euler totient, every element $g \in \mathbb{Z}_N^*$ satisfies

$$g^{\phi(N)} \equiv 1 \bmod N \tag{23}$$

Since $N = p \cdot q$, we have:

$$\phi(N) = (p-1)(q-1) = pq - p - q + 1 = N - (p+q-1) \tag{24}$$

Therefore:

$$g^{N-1} = g^{\phi(N)} \cdot g^{(p+q-2)} \equiv g^{(p+q-2)} \bmod N \tag{25}$$

Since p and q are distinct odd primes, $p-1$ and $q-1$ are both even, so $\gcd(p-1, q-1)$ is even. Hence $\gcd(p-1, q-1)/2$ is an integer. We use the Carmichael function $\lambda(N) = \operatorname{lcm}(p-1, q-1) = (p-1)(q-1)/\gcd(p-1, q-1)$. Then

$$\begin{aligned} \frac{\phi(N)/2}{\lambda(N)} &= \frac{(p-1)(q-1)/2}{(p-1)(q-1)/\gcd(p-1, q-1)} \\ &= \frac{\gcd(p-1, q-1)}{2} \in \mathbb{Z}, \end{aligned} \tag{26}$$

so $\lambda(N) \mid \phi(N)/2$, and hence $g^{\phi(N)/2} \equiv 1 \pmod N$ for every $g \in \mathbb{Z}_N^*$. Let $x := g^{(N-1)/2} \bmod N$ and $d := (p+q-2)/2$. Since $(N-1)/2 = \phi(N)/2 + d$ and $g^{\phi(N)/2} \equiv 1 \pmod N$, we obtain

$$x = g^{(N-1)/2} = g^{\phi(N)/2} \cdot g^d \equiv g^d \pmod N, \tag{27}$$

so $x = g^d \bmod N$. Since $p, q \approx 2^{n/2}$, we have $d \approx 2^{n/2}$, so d is small compared to N. What matters is its size relative to the order r of g, not to the order $\phi(N)$ of the whole group, because the algorithm only ever sees the cyclic group generated by g. We call d a *short* discrete logarithm if it is much smaller than r. The precise condition the algorithm needs is stated in Section A.5. For a random g it holds with overwhelming probability, because r is then within a small factor of $\phi(N) = \Theta(2^n)$.

This reformulation allows us to express the problem of period finding for factoring into the search of a short discrete log d. More specifically, x can be easily classically computed, as it is the exponentiation of a group element g to the power $(N-1)/2$ which is known. Once d is recovered using Shor's algorithm,

one can recover p and q by solving the quadratic equation $N = q(2d+2-q)$. Unlike the ordinary DLP case where $G = \mathbb{Z}_{p-1}^2$ is built from the known group order $p-1$, the group order $\phi(N)$ is unknown here (factoring N is the whole point). The registers are therefore built from a public upper bound on d's bit-length rather than the exact group order. This distinction is precisely why the two-register Ekerå–Håstad construction of Section A.5 looks different from the ordinary DLP circuit, where each register is sized directly from the known order $p-1$, up to the usual binary padding. Please note that with this reduction, Shor's algorithm for factoring is now in the two-dimensional set-up, namely we look for the generator $(d, 1)$ instead of the order r.

We can summarize the reduction from factoring to a short discrete-log problem as follows:

$$\underbrace{x = g^{(N-1)/2} \bmod N}_{\text{classically computable}} \;\Rightarrow\; \underbrace{d = \log_g x}_{\substack{\text{short DLP} \\ d \text{ of } \approx n/2 \text{ bits}}} \;\Rightarrow\; \underbrace{p + q = 2d + 2}_{\substack{\text{factor } N \text{ via} \\ q(2d+2-q)=N}} \tag{28}$$

We can now use the same algorithm to tackle the factoring problem and the discrete logarithm problem.

On the periodicity of the oracle f for two-dimensional factoring. The function $f(a, b) = g^{a-bd} \bmod N$ takes the same value on two pairs (a_1, b_1) and (a_2, b_2) if and only if $g^{a_1-b_1 d} \equiv g^{a_2-b_2 d} \pmod N$. Since g has multiplicative order r, this holds if and only if

$$a_1 - b_1 d \equiv a_2 - b_2 d \pmod r. \tag{29}$$

The oracle f is therefore periodic in the (a, b)-plane in two independent directions. First, shifting a by r with b fixed leaves f unchanged:

$$f(a + r,\, b) = g^{a+r-bd} = g^{a-bd} \cdot g^r = g^{a-bd} = f(a, b), \tag{30}$$

so f has period r in the a-direction. Second, shifting (a, b) by $(d, 1)$ also leaves f unchanged:

$$f(a + d,\, b + 1) = g^{(a+d)-(b+1)d} = g^{a-bd} = f(a, b), \tag{31}$$

so f is also periodic along the direction $(d, 1)$ in the (a, b)-plane. The first periodicity comes from $g^r = 1$. The second comes from $x = g^d$, that is, from d itself. Together they generate the period lattice

$$\begin{aligned} H &= \big\{(a, b) \in \mathbb{Z}^2 : a \equiv bd \pmod r\big\} \\ &= \big\langle (r, 0),\, (d, 1) \big\rangle \leq \mathbb{Z}^2, \end{aligned} \tag{32}$$

and f is constant and separating on its cosets. This H plays the role of the hidden subgroup of the discrete-logarithm instance of Section A.1, with $\mathbb{Z}^2$ in place of $\mathbb{Z}_{p-1}^2$ because the order r is unknown. Its image in $\mathbb{Z}_r \times \mathbb{Z}_r$ is the cyclic subgroup $\langle (d, 1) \rangle$ of order r. Recovering H determines d only modulo r, since $(d, 1)$ and $(d + r, 1)$ generate the same lattice. The reduction therefore needs $d < r$, which the hypothesis of Section A.5 guarantees.

A.5 Reducing the Input Register Size

The exponent register has two costs, and which one this section reduces depends on the oracle. With recycling (Section A.3.0.v), the number of qubits held live at any one time is independent of how long the exponent is. What shrinks below is then the length itself, and with it the number of windows the recycling loop has to step through, and so the number of controlled multiplications the oracle performs. In that setting the reductions of this section are depth optimizations. Recycling is not always available. The residue-arithmetic oracle of [10, 11] is incompatible with it [11, Sect. 1], so the whole exponent register must be held for the entire run. There the reductions of this section save qubits. This is the regime of the RSA-2048 pipeline of Section A.9.

As we have seen, the Ekerå–Håstad reduction recasts factoring as the search for the short logarithm $d = (p+q-2)/2$. Since $p, q \approx 2^{n/2}$, we have $d \approx 2^{n/2}$, so d has bit-length $m \approx n/2$, roughly half the bit-length of the group order $\phi(N) \approx 2^n$. The algorithm of [8] exploits this by trading a single run for several runs combined through classical lattice reduction. Its two exponent registers together hold $n_e = \left(\frac{1}{2} + \frac{1}{s}\right) n$ bits, for an integer parameter $s \geq 2$ (the hypothesis below excludes $s = 1$ for RSA moduli). This is n bits at $s = 2$, half of Shor's $2n$. As s grows it approaches $n/2$ bits, a factor of four below Shor, at the price of more runs.

i Two index registers and a shorter exponent. The algorithm uses two exponent registers rather than the single one of order finding. They have $\tau_s + m$ and τ_s qubits, with $\tau_s \approx m/s$. Write g for the chosen generator and $x = g^d$ for the target. The quantum stage prepares a uniform superposition over both registers and evaluates

$$\begin{aligned} |a\rangle\,|b\rangle\,|0\rangle \;&\longmapsto\; |a\rangle\,|b\rangle\,\big|\, g^a x^{-b} \bmod N\big\rangle \\ &= |a\rangle\,|b\rangle\,\big|\, g^{a-bd} \bmod N\big\rangle\,, \end{aligned} \tag{33}$$

with $0 \leq a < 2^{\tau_s+m}$ and $0 \leq b < 2^{\tau_s}$. Since $m \approx n/2$ once the short-DLP recasting has been applied and $\tau_s \approx m/s$, the first register has $\tau_s + m \approx \left(\frac{1}{2} + \frac{1}{2s}\right) n$ qubits and the second $\tau_s \approx n/(2s)$. Increasing s shrinks τ_s, and therefore the register, at the price of requiring more runs. Specifically, this approach trades the exact single-shot recovery of Shor's QFT for a noisier measurement that still constrains d. To fully recover d from those noisy linear constraints, one now requires multiple measurements to have a suitable, solvable system. Counting both registers, the exponent has $n_e = (\tau_s + m) + \tau_s = \left(\frac{1}{2} + \frac{1}{s}\right) n$ bits.

ii The hypothesis on the order of g. Let m be the smallest integer with $0 < d < 2^m$, so $m \approx n/2$ here. Ekerå and Håstad require the order r of g to

satisfy

$$r \;\geq\; 2^{\tau_s+m} + 2^{\tau_s}\, d \tag{34}$$

[8, Theorem 1]. It guarantees that the exponent $a-bd$ is never reduced modulo r for any (a, b) in the registers, and the analysis below relies on this. Without it, the algorithm only determines $d \bmod r$. The order r need not be known. For factoring, write $r = \phi(N)/t$. Since $d < 2^m$, it suffices that $r \geq 2^{\tau_s+m+1}$, and with $\phi(N) \approx 2^n$ this amounts to a bound of roughly $2^{n-\tau_s-m}$ on the index t. For random p, q and a random g the index is small, and Ekerå and Håstad show that the bound holds with overwhelming probability for every $s > 1$ [8, Sect. 5.2.4]. It fails for $s = 1$, since 2^{τ_s+m} then reaches about 2^n, which exceeds $N > r$.

iii Where the QFT peaks in the two-dimensional setting. In the one-dimensional order finding of Section A.3, one register produced one measured frequency, sharply peaked at a near-multiple of $2^{n_e}/r$, from which continued fractions recovered r. The two-dimensional algorithm behaves differently. Under hypothesis (34) the registers hold less than one period of f, so no value of j is singled out on its own. All the information about d sits in a single approximate alignment between j and k, which we now derive following [8, Sect. 4.3].

After the oracle, the joint state is

$$|\psi\rangle = \frac{1}{\sqrt{2^{\tau_s+m}\cdot 2^{\tau_s}}} \sum_{a=0}^{2^{\tau_s+m}-1} \sum_{b=0}^{2^{\tau_s}-1} |a\rangle|b\rangle|g^{a-bd} \bmod N\rangle. \tag{35}$$

Write $e = a - bd$ for the exponent. For (a, b) in the registers, e ranges over an interval of fewer than r integers, by (34). Distinct values of e therefore give distinct group elements g^e, so the output register determines e exactly, with no reduction modulo r. Applying $\mathrm{QFT}^{-1}_{2^{\tau_s+m}}$ to the first register and $\mathrm{QFT}^{-1}_{2^{\tau_s}}$ to the second, the state $|j, k, g^e\rangle$ is observed with probability

$$P(j,k,e) = \frac{1}{2^{2(2\tau_s+m)}} \Big| \sum_{b\in B_e} e^{2\pi i\,((e+bd)j+2^m bk)/2^{\tau_s+m}} \Big|^2, \tag{36}$$

where $B_e = \{0 \leq b < 2^{\tau_s} : 0 \leq e + bd < 2^{\tau_s+m}\}$ collects the pairs with $a = e + bd$ in range. Write $T_e = |B_e|$. The two phase factors have been combined into one, since $k/2^{\tau_s} = k \cdot 2^m/2^{\tau_s+m}$. Extracting the factor $e^{2\pi i\, ej/2^{\tau_s+m}}$, which has modulus one, leaves a sum over b alone,

$$P(j,k,e) = \frac{1}{2^{2(2\tau_s+m)}} \Big| \sum_{b\in B_e} e^{2\pi i\, b\,\{dj+2^m k\}_{2^{\tau_s+m}}/2^{\tau_s+m}} \Big|^2. \tag{37}$$

Here $\{\cdot\}_{2^{\tau_s+m}}$ denotes the centered residue in $[-2^{\tau_s+m-1}, 2^{\tau_s+m-1})$. Replacing $dj + 2^m k$ by its residue shifts each phase by a multiple of 2π and changes nothing.

This is a geometric series in b with per-step phase $2\pi\{dj + 2^m k\}_{2^{\tau_s+m}}/2^{\tau_s+m}$. It is large only when that phase is small, that is when the integer $dj + 2^m k$ is close to a multiple of 2^{τ_s+m}. This is a linear relation between the unknown d and the measured j, k. Because $2^m k$ takes only values that are multiples of 2^m, the frequency k aligns with the high-order part of dj. The remainder $dj \bmod 2^m$ is the part of the relation that constrains d. This alignment is only ever approximate, and it is the source of the noise in the two-dimensional outcome.

iv The good pair condition. That approximate alignment has a precise geometric meaning. The T_e phasors summed in (36) spread across an arc of the unit circle whose width grows with $|\{dj+2^m k\}_{2^{\tau_s+m}}|$. When the spread is too large they cancel, giving a low-probability outcome that carries no usable information about d. Ekerå and Håstad [8] show (Claim 1) that when every phase angle satisfies $|\theta_i| \leq \pi/4$, the squared norm of a sum of T phasors is at least $T^2/2$, half of its maximal value. This is their $\pm\pi/4$ constructive-interference condition. Shifting the summation index to $b - 2^{\tau_s-1}$ only multiplies the sum by a phase, and centers the angles around zero.

Call a pair (j, k) *good* when

$$\big|\{\, dj + 2^m k \,\}_{2^{\tau_s+m}}\big| \leq 2^{m-2}. \tag{38}$$

Since $|b - 2^{\tau_s-1}| \leq 2^{\tau_s-1}$, this bound caps every centered angle at $2^{\tau_s-1} \cdot 2\pi \cdot 2^{m-2}/2^{\tau_s+m} = \pi/4$. The probability of observing a given good pair together with a given e is therefore at least $T_e^2/(2 \cdot 2^{2(2\tau_s+m)})$. The values T_e sum to $2^{2\tau_s+m}$ over fewer than 2^{τ_s+m+1} values of e, so the Cauchy–Schwarz inequality gives $\sum_e T_e^2 \geq 2^{3\tau_s+m-1}$. Summing over e, each good pair is observed with probability at least $2^{-m-\tau_s-2}$ [8, Lemma 2]. At least 2^{τ_s+m-1} values of j admit a good k [8, Lemma 1], so a single shot yields a good pair with probability at least $1/8$. A good pair is a usable linear constraint on d. Its j acts as a known random multiplier, and its k records the high-order bits of dj, fixing where dj wraps modulo 2^m. Figure 9 illustrates the condition on a toy instance.

The pair (j, k) is computed exactly, so $\epsilon = 0$. The read-out of Ekerå and Håstad cannot tell a good pair from a bad one, since the test (38) involves the unknown d. Every pair is therefore accepted and read as the constraint (38). For a bad pair that constraint is false, so under Definition A.5 a bad pair is a corruption, not an abort. This is a $(0, \delta)$-noisy HSP instance with $\delta_\perp = 0$ and $\delta = \delta_\times \leq 7/8$. The corruptions are caught one level up, on a whole attempt. Ekerå and Håstad run $8s$ shots, apply the lattice step below to every subset of s pairs, and accept a candidate only if it satisfies $x = g^d$ [8, Sect. 4.5]. The check is exact, so a wrong d is never output. A corrupted subset only costs another subset, or another attempt.

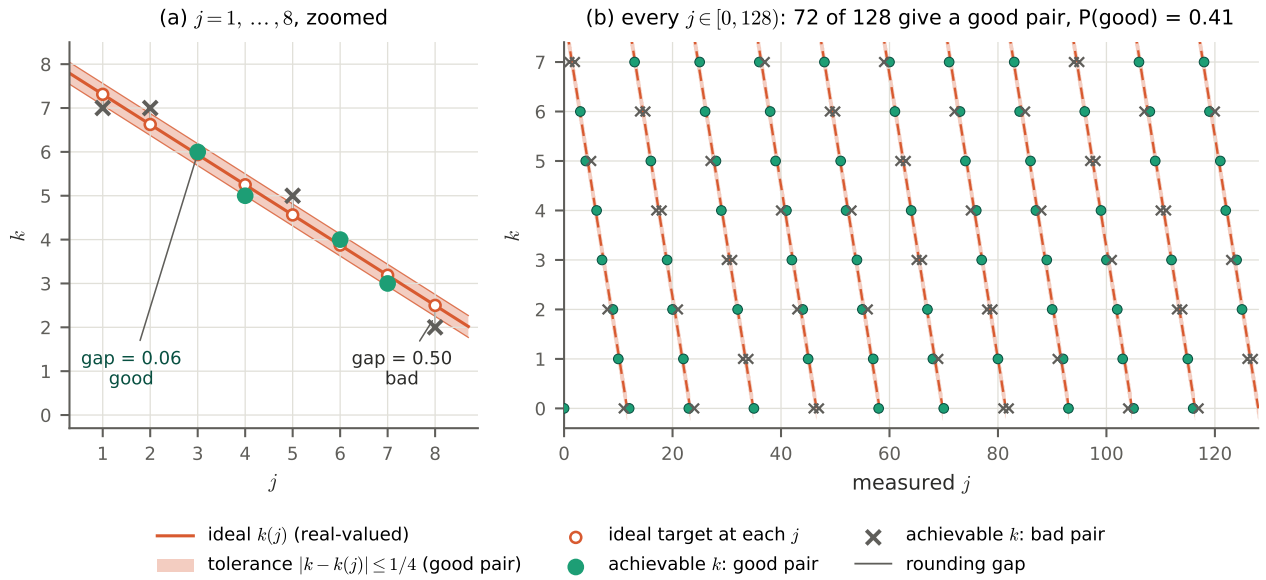
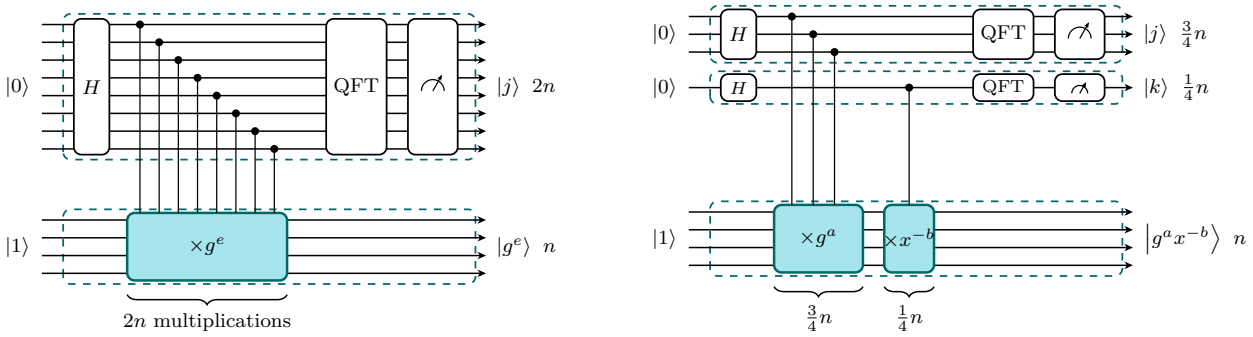


Figure 9: Illustration of the good-pair condition (38) of Section A.5.0.iv, for a toy instance with $\tau_s = 3$, $m = 4$ and $d = 11$, so that $2^{\tau_s+m} = 128$ and k takes $2^{\tau_s} = 8$ values. Hypothesis (34) requires $r \ge 2^{\tau_s+m} + 2^{\tau_s} d = 216$, which holds for instance for $g = 2$ in $\mathbb{Z}^*_{227}$, of order $r = 226$. The condition itself does not depend on r. For each j, the curve gives the real-valued $k(j) = -\frac{d}{2^m} j \bmod 2^{\tau_s}$ that would make $dj + 2^m k$ an exact multiple of 2^{τ_s+m}. Condition (38) is equivalent to $|k - k(j)| \le 2^{m-2}/2^m = 1/4$, the shaded band, so a pair is good when its integer k falls inside it. **(a)** The first 8 values of j, zoomed in. Filled circles are good pairs, crosses bad ones, and the thin line is the rounding gap between $k(j)$ and the nearest integer. **(b)** Every $j \in [0, 2^{\tau_s+m})$, each wrap of the curve back to $k = 2^{\tau_s}$ starting a new period. For 72 of the 128 values of j, the nearest integer k gives a good pair.



(a) Shor [2]

(b) Ekerå–Håstad [8], drawn for $s = 2$

Figure 10: Comparison of the input registers, drawn to a common scale in which one wire stands for $n/4$ qubits. (a): Shor's original circuit uses a single $2n$-qubit index register evaluating g^e. (b): the Ekerå–Håstad circuit uses two index registers of $n/2 + n/(2s)$ and $n/(2s)$ qubits, evaluating $g^a x^{-b}$ in superposition, shown here for $s = 2$. The output register holds n qubits in both. Measuring it is optional (Figure 4) and is not drawn. All arithmetic is modulo N, and the $\bmod N$ is omitted inside the boxes.

v Lattice-based recovery of d. A single good pair does not determine d. The short logarithm is recovered by combining s of them, $(j_1, k_1), \ldots, (j_s, k_s)$, through lattice reduction. Consider the $(s+1)$-dimensional integer lattice Λ generated by the rows of

$$\begin{bmatrix} j_1 & j_2 & \cdots & j_s & 1 \\ 2^{\tau_s+m} & 0 & \cdots & 0 & 0 \\ 0 & 2^{\tau_s+m} & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 2^{\tau_s+m} & 0 \end{bmatrix}, \quad (39)$$

together with the target vector

$$\mathbf{v} = \left(\{-2^m k_1\}_{2^{\tau_s+m}}, \ldots, \{-2^m k_s\}_{2^{\tau_s+m}}, 0\right). \quad (40)$$

By construction, the lattice contains the vector

$$\mathbf{u} = \big(\{dj_1\}_{2^{\tau_s+m}} + m_1 2^{\tau_s+m},\ \ldots,\ \{dj_s\}_{2^{\tau_s+m}} + m_s 2^{\tau_s+m},\ d\big) \tag{41}$$

where $m_1, \ldots, m_s$ are the unknown wrap counts, whose last component is exactly d, and Ekerå and Håstad derive that the good-pair condition guarantees $|\mathbf{u} - \mathbf{v}| < \sqrt{s/4+1} \cdot 2^m$. Recovering d therefore amounts to a bounded-distance search. One enumerates the lattice points within this radius of $\mathbf{v}$ and returns the last component that satisfies $x = g^d$. Since Λ has determinant $\approx 2^{m(s+1)}$ in dimension $s+1$, its shortest vector has norm $\approx 2^m$. One shows that, with high probability, only a constant number of lattice points (depending on s) fall within the search radius. The search is thus efficient, and the whole post-processing runs in classical polynomial time.

This is the same design philosophy seen throughout the arithmetic computations of Shor's algorithm, shifting work from the quantum to the classical world. Here the single-shot continued-fraction recovery of the order-finding case is replaced by a multi-shot lattice computation. It tolerates a smaller input register, at the cost of a constant number of additional runs and a polynomial-time classical lattice search.

An illustration of the Ekerå–Håstad circuit that allows the reduction of the input register is depicted in Figure 10.

A.6 Reducing the Output Register Size

The previous optimization shrank the input register but left the output register untouched. It still stores the full n-bit value $g^a x^{-b} \bmod N$, which for RSA dominates the qubit budget. Counting only the registers that are measured, Ekerå–Håstad uses $\big(\frac{1}{2} + \frac{1}{s}\big)n$ input qubits and n output qubits, which approaches $\frac{3}{2}n$ as s grows. Both counts leave out the workspace of the arithmetic, which Section A.3 counts separately. May and Schlieper [9] observed that the output register is far larger than period finding needs, and can be compressed to as few as $t = 1$ qubit. Their result holds in an *oracle model*. It assumes a circuit $U_{h\circ f}$ that writes the hash $h(f(a,b))$ directly into t qubits. Computing f first and hashing the result afterwards is not enough, since that needs the n qubits hashing was meant to save [9, Sect. 1]. How to build $U_{h\circ f}$ was left open, and Section A.7 describes how it was later solved.

i Compression is periodicity-preserving. The key observation is that period finding never uses the value of $f(a,b) = g^a x^{-b} \bmod N$ itself. It only uses the fact that f is periodic. Composing f with any function h preserves that periodicity, since $f(a,b) = f(a+d, b+1)$ immediately gives

$$h\big(f(a,b)\big) = h\big(f(a+d,b+1)\big), \quad \text{for all } a,b. \tag{42}$$

In particular one may take h to hash the output all the way down to a single bit. Doing so introduces many collisions $h(f(a,b)) = h(f(a',b'))$ between distinct cosets of H, that is, with $(a-a') - (b-b')d \not\equiv 0 \pmod r$. Remarkably, these collisions do not corrupt the period signal. They only add weight to the useless outcome.

ii May–Schlieper main result. Let $\mathcal{H}_t = \{\, h : \mathbb{Z}_N \to \{0,1\}^t \,\}$ be a *universal* family of hash functions in the sense of May and Schlieper [9, Def. 1], meaning that any two distinct inputs collide with probability *exactly* 2^{-t}:

$$\Pr_{h\in\mathcal{H}_t}[h(z_1) = h(z_2)] = 2^{-t} \qquad \text{for all } z_1 \neq z_2. \tag{43}$$

Suppose we replace the oracle f by its hashed version $h \circ f$, drawing a fresh h uniformly from $\mathcal{H}_t$ at each run. Write $p(y)$ for the probability of measuring the frequency y with the original f, and $p_h(y)$ for the probability with $h \circ f$, taken over both the measurement and the random choice of h. May and Schlieper prove that

$$p_h(y) = \big(1 - 2^{-t}\big)\, p(y) \qquad \text{for every } y \neq 0 \tag{44}$$

[9, Thm. 7], provided the circuit meets their cancellation criterion [9, Eq. (14)]. Write the state before measurement as $\sum_y \sum_z w_{y,z} \,|y\rangle\,|z\rangle$, with z ranging over the values of f. The criterion asks that $\sum_z w_{y,z} = 0$ for every $y \neq 0$. It holds for Shor's circuit and for its Ekerå–Håstad variant. The sum $\sum_z w_{y,z}$ is the amplitude the circuit would give y if f were constant, and a QFT of a uniform superposition puts all of that amplitude on $y = 0$. Every informative outcome $y \neq 0$ therefore keeps its probability up to the same factor $1 - 2^{-t}$, on average over h. Since both distributions sum to one, the removed probability reappears on the outcome $y = 0$:

$$p_h(0) = p(0) + 2^{-t} \sum_{y\neq 0} p(y). \tag{45}$$

On average over h, collisions move weight from every informative outcome to the bin $y = 0$ at the same rate 2^{-t}. For a fixed h the rate depends on y [9, Eq. (13) and Fig. 11b]. This is depicted in Figure 11.

iii Consequences. Two conclusions follow directly. First, the classical post-processing is unchanged. Averaged over the random choice of h, the outcomes $y \neq 0$ keep their relative probabilities, so the lattice-based recovery of the previous section still applies. Second, the cost is only a constant factor in runs. With a t-bit hash a nonzero outcome appears with probability $(1 - 2^{-t})(1 - p(0))$, so the expected number of measurements grows by a factor $1/(1 - 2^{-t})$. For the extreme choice $t = 1$ this is

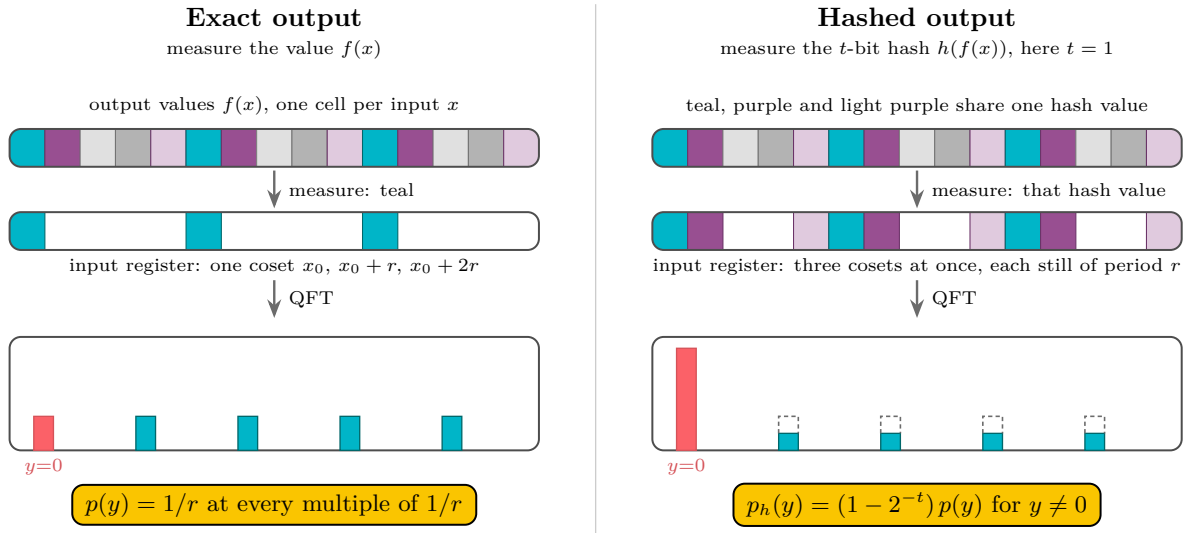


Figure 11: How hashing the output creates collisions (Equation (45)), for a toy function of period $r = 5$ on 15 inputs, in one dimension as in Section A.3. The same picture applies to each coset of the two-dimensional lattice of Section A.5. In the top bars, each cell is one input x and its color is the value $f(x)$, so colors repeat every r cells. Left: measuring $f(x)$ collapses the input register to the one coset of the measured value, and the QFT gives equal peaks $p(y) = 1/r$ at the multiples of $1/r$, including the uninformative $y = 0$. Right: a t-bit hash ($t = 1$ shown) gives the same value to about $r/2^t$ cosets, here three, so the input register collapses to all of them at once. Each is still of period r, so the peaks stay at the same positions. Here $p_h(y)$ is the outcome probability with the hashed oracle, averaged over the random choice of h. Each informative peak is scaled by $1 - 2^{-t}$ (dashed: exact height), and the removed weight goes to $y = 0$. A large t recovers exact Shor, while $t = 1$ doubles the number of runs and saves n output qubits.

a factor of two, that is, twice as many shots in exchange for discarding all but one output qubit. Applied to Ekerå–Håstad in the oracle model, this lowers the measured registers for factoring n-bit RSA from about $\frac{3}{2}n$ to $\big(\frac{1}{2} + o(1)\big)n$ qubits as s grows, workspace excluded.

The oracle f is exact ($\epsilon = 0$). The read-out R draws a fresh hash h from $\mathcal{H}_t$ for each run and measures $h \circ f$. Hashing adds no corruption. Under Definition A.5, δ is a probability over the whole shot, including the draw of h. What matters is therefore the outcome distribution averaged over h. By Theorem 7 of [9], that distribution, conditioned on $y \neq 0$, is exactly the one without hashing, so the conditional corruption rate is unchanged. For an exact transform there is a stronger reason, which holds for every h. Two elements of the same coset of H share the value of f, hence of $h \circ f$, so $h \circ f$ is constant on the cosets of H. Its spectrum therefore stays on the subgroup orthogonal to H, and no outcome can carry a false constraint. For a fixed h and a power-of-two transform no such per-h statement is available, since the rescaling then depends on y. What hashing adds is aborts. The abort rate of Definition A.5 is $\delta_\perp = p_h(0) = p(0) + 2^{-t}\big(1 - p(0)\big)$, which is at least 2^{-t}. The read-out without hashing already discards $y = 0$ with probability $p(0)$, so the abort rate the hash adds is $2^{-t}\big(1 - p(0)\big) \leq 2^{-t}$. This added rate is the entry of Table 3.

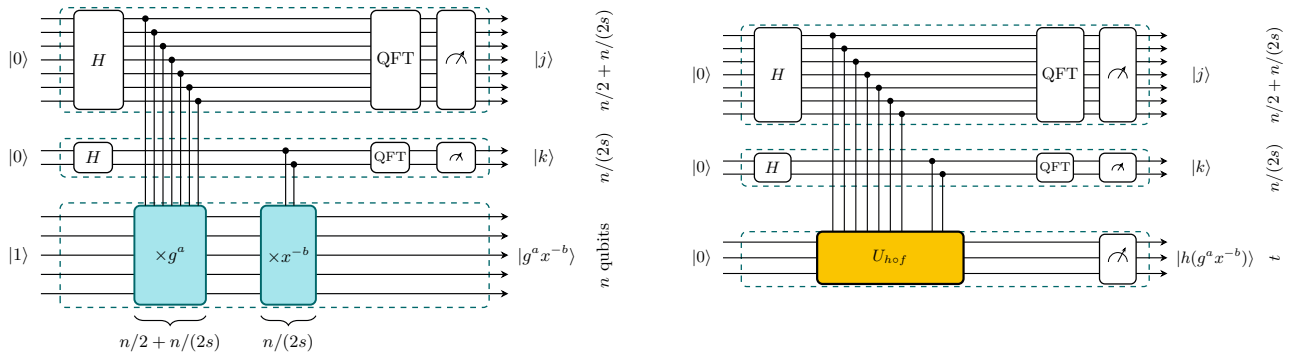
The effect of this compression on the output register is illustrated in Figure 12, contrasting the full n-qubit measured register of Ekerå–Håstad with the t-qubit hashed register of May-Schlieper.

iv The remaining obstacle. As noted at the start of this section, this bound holds only in the oracle model, which assumes black-box access to $U_{h \circ f}$. Computing $g^a x^{-b} \bmod N$ first and hashing the result afterwards is not enough. That would reintroduce the n output qubits one is trying to eliminate. Computing $h(g^a x^{-b} \bmod N)$ directly, without ever materializing the n-bit intermediate value, is left open by May and Schlieper and is precisely the problem solved by the residue-arithmetic construction of the next section.

A.7 Making the Oracle Function Space Efficient

The May-Schlieper compression leaves one obstacle: computing $h(g^a x^{-b} \bmod N)$ without ever forming the n-bit value $g^a x^{-b} \bmod N$ in an intermediate register. Chevignard, Fouque and Schrottenloher [10] solved it, and Gidney [11] later improved the construction. The key idea is to carry out the entire modular exponentiation in a *residue number system* (RNS), where every intermediate value fits in $O(\log \log N)$ qubits. Only the most significant bits of the result are then extracted, through an approximate accumulation.

As in the description of Shor's algorithm in Section A.3, we present the residue arithmetic, and the approximate period finding that follows it, for a one-



(a) Ekerå–Håstad [8]

(b) May–Schlieper, oracle model [9]

Figure 12: Output-register compression, drawn to a common scale in which one wire stands for $n/4$ qubits. (a): in Ekerå–Håstad the measured output register holds the full n-bit value $g^a x^{-b} \bmod N$. (b): in the oracle model of May and Schlieper, a single circuit $U_{h\circ f}$ writes $h(g^a x^{-b} \bmod N)$ into $t \ll n$ measured qubits, at the cost of a factor $1/(1-2^{-t})$ in the number of runs. All arithmetic is modulo N, and the $\bmod N$ is omitted inside the boxes.

dimensional exponent e and period r. This keeps the notation light. Nothing in this construction or the next depends on the dimension. Every statement below applies unchanged to the two-register Ekerå–Håstad setting, with e replaced by the pair (a, b) and $H = \langle r \rangle$ replaced by the lattice $H = \langle (r, 0), (d, 1) \rangle$ of Section A.4. This is the setting actually used once the reductions of Sections A.5 and A.6 are combined with the oracle below.

i Residue arithmetic. Fix a set P of ℓ-bit primes and let $L = \prod_{p \in P} p$. The residue number system represents an integer $z < L$ by its residues $(z \bmod p)_{p \in P}$, and performs additions and multiplications independently modulo each prime, reconstructing the result through the Chinese remainder theorem (CRT). By the prime number theorem, the primes can be taken of size $\ell = \Theta(\log \log L)$ to have enough primes to supply P: there are $\sim 2^\ell / \ell \ln 2$ primes of size ℓ, forcing $2^\ell \geq n_e n$. Each residue occupies only $O(\log \log N)$ qubits. The modular exponentiation

$$V = g^e \bmod N = \left(\prod_{k=0}^{n_e - 1} M_k^{e_k} \right) \bmod N, \quad \text{with } M_k = g^{2^k} \bmod N, \tag{46}$$

[9]

is not reduced modulo N during the quantum computation. In a residue number system, a reduction modulo N is expensive, because it would require reconstructing the value (or comparing it against N) inside the circuit. That defeats the purpose of the RNS. Its additions and multiplications are cheap precisely because they act componentwise modulo the fixed primes $p \in P$. The circuit therefore accumulates the unreduced product $\prod_k M_k^{e_k}$ modulo L and performs a single reduction modulo N only at the end. This is safe provided the accumulator never wraps modulo L. Each $M_k < N$, and at most n_e factors are multiplied, so the unreduced product stays below N^{n_e}. Hence, as in [10], one requires

$$L \geq N^{n_e}. \tag{47}$$

This guarantees that the product never exceeds L. Write $\bar{V} = \prod_k M_k^{e_k}$ for this unreduced product, so that $V = \bar{V} \bmod N$. The circuit holds $\bar{V}$ through its residues $r_j = \bar{V} \bmod p_j$. Since $\bar{V} < L$, the Chinese remainder theorem recovers it from them, using the CRT contribution factors $u_{p_j} = (L/p_j) \cdot ((L/p_j)^{-1} \bmod p_j)$. Writing $r_{j,k}$ for the k-th bit of r_j, with j ranging over $0, \ldots, |P| - 1$ and k over $0, \ldots, \ell - 1$, this gives the exact identity [11, Eq. (13)]

$$\begin{aligned} V &= \Big(\sum_j r_j \, u_{p_j} \Big) \bmod L \bmod N \\ &= \Big(\sum_j \sum_k r_{j,k} \, u_{p_j} \, 2^k \Big) \bmod L \bmod N. \end{aligned}$$

This turns a product modulo N into a sum, which is what makes the subsequent approximation stable. An error introduced early is no longer amplified by later multiplications. The coefficients u_{p_j} satisfy $u_{p_j} \equiv 1 \pmod{p_j}$ and $u_{p_j} \equiv 0 \pmod{p_i}$ for $i \neq j$, by definition of the modular inverse. Hence $\sum_i r_i u_{p_i} \equiv r_j \pmod{p_j}$ for each j, which recovers $\bar{V} \bmod p_j$ from the residues. This is precisely the CRT reconstruction.

The circuit cannot afford the outer reduction modulo L either. It therefore reduces each term separately and accumulates modulo N only,

$$V \approx \Big(\sum_j \sum_k r_{j,k} \left[(u_{p_j} 2^k) \bmod L \bmod N \right] \Big) \bmod N. \tag{48}$$

This is not an identity. After its own reduction modulo L, each term is below L, so the sum of the reduced terms wraps modulo L fewer than $A = |P| \cdot \ell$ times. Each wrap that the right-hand side ignores shifts it by $L \bmod N$. The two sides of (48) therefore differ by a modular distance of at most $A \, \Delta_N(L)$ [11, Eq. (19)].

Here $\Delta_N(z) = \min_{k\in\mathbb{Z}} |z - kN|$ denotes the distance from z to the nearest multiple of N, i.e. the size of the centered residue $\{z\}_N \in [-N/2, N/2)$. To keep this error at the truncation scale, a second condition on L is required,

$$\Delta_N(L) \;<\; N\,2^{-\kappa}, \tag{49}$$

where κ denotes the number of top bits kept (see hereafter). The primes in P are ours to choose, and Gidney finds a suitable L by random search over prime sets, under a stated conjecture [11, Assumption 1].

ii Extracting the most significant bits. Rather than compute V in full, the algorithm keeps only its top κ bits. Additions are truncated: to add an offset c one instead adds $(c \bmod N) \gg \bar{\kappa}$ with $\bar{\kappa} = \log N - \kappa$, and the accumulator is operated modulo $N \gg \bar{\kappa}$. Each truncated addition drops the low $\bar{\kappa}$ bits, perturbing the result by an absolute modular distance of at most $O(2^{\bar{\kappa}}) = O(N \cdot 2^{-\kappa})$, equivalently, by a fractional deviation $O(2^{-\kappa})$ of the modulus. Since deviations accumulate linearly over the $|P| \cdot \ell$ additions, and the wrap error of (48) is below $A\,N2^{-\kappa}$ by (49), the total deviation of the approximation $\tilde{V} \approx V \gg \bar{\kappa}$ is

$$\Delta_N\big(V - (\tilde{V} \ll \bar{\kappa})\big) \le \frac{\epsilon}{2}\,N, \qquad \epsilon = O\big(|P| \cdot \ell \cdot 2^{-\kappa}\big)\,. \tag{50}$$

The accumulator holds only the top κ bits of the result, so its raw value is about $2^{\bar{\kappa}}$ times smaller than V. To compare the two on the same scale, we write

$$\hat{V}(e) := \tilde{V}(e) \ll \bar{\kappa} = 2^{\bar{\kappa}}\,\tilde{V}(e) \tag{51}$$

for the accumulator content scaled back to the size of N. Equation (50) then reads $\Delta_N\big(V(e) - \hat{V}(e)\big) \le \frac{\epsilon}{2}N$.

Equation (50) is stated with the same $\epsilon/2$ normalization as condition (i) of Definition A.6. So $\hat{V}$ instantiates that condition with exactly this ϵ, and no rescaling is needed. The bound is only informative for $\epsilon < 1$: since Δ_N never exceeds $N/2$, any claimed bound of $\frac{\epsilon}{2}N \ge N/2$ holds. The choice of κ in the success-rate analysis keeps ϵ far below this threshold, driving it to an arbitrarily small constant, not merely below 1. The approximation reduces the modular exponentiation to a κ-bit dot product against classically precomputed constants. These constants are $C_{j,k} = \big[((u_{p_j} 2^k) \bmod L \bmod N) \gg \bar{\kappa}\big] \bmod (N \gg \bar{\kappa})$, and the dot product reads

$$\tilde{V} = \left(\sum_{j=0}^{|P|-1} \sum_{k=0}^{\ell-1} r_{j,k}\, C_{j,k} \right) \bmod (N \gg \bar{\kappa}). \tag{52}$$

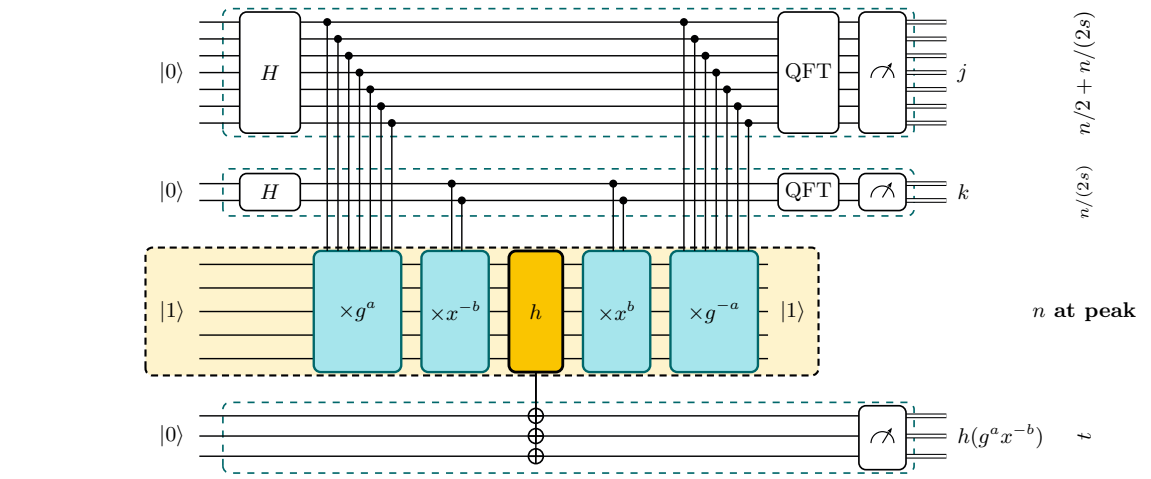
Every quantum register involved, apart from the input register itself, has size $O(\log\log N)$: the n-qubit intermediate value has been eliminated. Figure 14 summarizes the three steps on a toy example: residues, lane-wise arithmetic, and keeping only the top bits. The output is then hidden by superposition masking rather than by an explicit hash. The approximate value $\tilde{V}$ is added directly into the mask register, which is immediately measured. As a result, no n-qubit intermediate state ever accumulates, and no reverse exponentiation is needed to uncompute it. The two techniques share the same shape, fresh randomness drawn each run, then a measurement that reveals only part of the value, but not the same guarantee. May-Schlieper's h acts on the exact value $f(a, b)$, so its row of Table 3 has $\epsilon = 0$ and can only waste a shot, never corrupt one. Masking instead acts on $\tilde{V}$, which already carries the truncation drift ϵ from above, so it can corrupt a shot too. κ and the mask width S are accordingly two separate parameters, with κ fixing the drift ϵ that S then has to be chosen large enough to absorb, made precise in the success-rate paragraph below. We explain this technique in more detail below.

iii The prime basis and its parameters. The residue system is where the two principal cost parameters of the algorithm are set, and they act on the two axes independently. First, the *size* of the primes fixes the qubit count: each residue is $\ell = \Theta(\log\log N)$ qubits wide, so the space cost of the arithmetic is logarithmic in the size of N. Second, the *number* of primes fixes the amount of computation. The size promise $L = \prod_{p\in P} p \ge N^{n_e}$ combined with $\ell = \Theta(\log\log N)$ forces, via the prime number theorem,

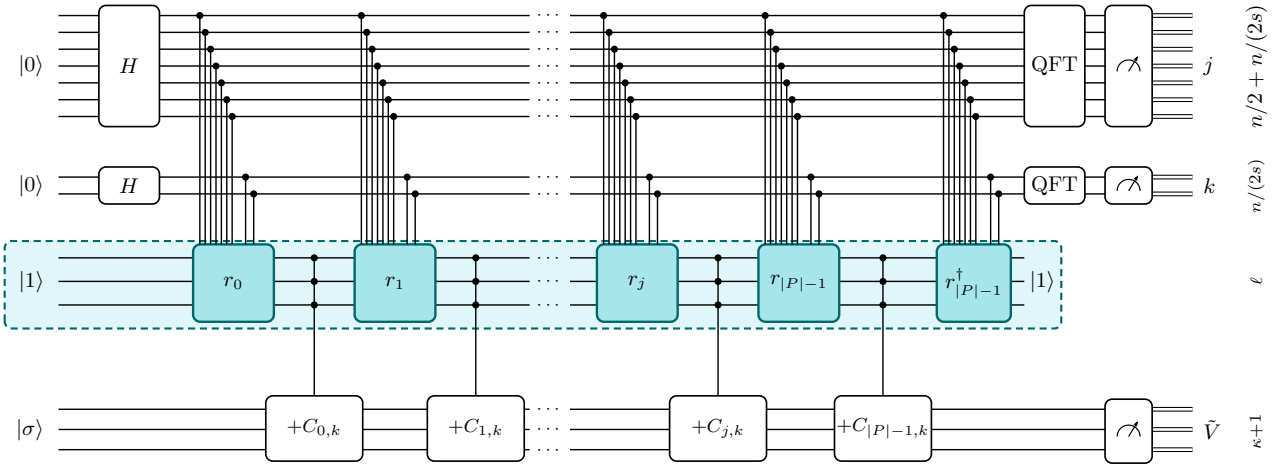
$$|P| = \Theta\!\left(\frac{\log L}{\log\log L}\right) = \Theta\!\left(\frac{n_e\, n}{\ell}\right), \tag{53}$$

and each prime demands its own controlled exponentiation to form the residue r_j. The number of these exponentiations therefore scales with $|P|$. Anything that lowers n_e, the number of input qubits, is beneficial. A smaller n_e shrinks the exponent, and with it L, $|P|$ and the number of exponentiations. This is precisely why combining the RNS with the Ekerå–Håstad reduction and with windowing is so effective. The Ekerå–Håstad reduction lowers n_e to $n_e = \frac{n}{2} + 2\lceil\frac{n}{2s}\rceil \approx \big(\frac{1}{2} + \frac{1}{s}\big)n$. This is also why increasing the parameter s yields a computation that is both smaller and faster [11].

This residue-based approach has a clear space-time trade-off. Rather than holding the full n-qubit intermediate value in memory, the algorithm reuses a single ℓ-qubit register to compute one residue at a time. Each contribution is accumulated into the mask, and the residue is uncomputed before moving on to the next prime. This lowers the working qubit count from $O(\log N)$ to $O(\log\log N)$, but at the price of more gate operations. The value must now be reconstructed from $|P| = \Theta(n_e n/\ell)$ sequential residue computations, each a small exponentiation in its own right. The construction therefore trades gates for space. Gidney estimates that the



(a) May–Schlieper, computed directly [9]



(b) Gidney [11]

Figure 13: Making the oracle space-efficient, both panels at the same scale. All arithmetic is modulo N, and the $\bmod N$ is omitted inside the boxes. (a) Computing the May–Schlieper oracle directly forms the full n-qubit value $g^a x^{-b} \bmod N$ in the highlighted register, hashes it, and uncomputes it, so n qubits are still needed at peak. (b) The residue-number-system construction of [10, 11] never forms that value. The residues r_j live in a single $\ell = O(\log n)$-qubit register. From the second box on, each box uncomputes the previous residue while computing the next one (Figure 15), and the last box uncomputes $r_{|P|-1}$. A box $+C_{j,k}$ on the accumulator adds the constant $C_{j,k}$ of Equation (52), controlled by the bit $r_{j,k}$ of the residue r_j. The $(\kappa{+}1)$-qubit accumulator starts in the mask state $|\sigma\rangle$ of Section A.8 and is measured at the end. Dots mark the primes omitted between r_1 and r_j.

original construction of [10] pays roughly a thousand times more Toffoli gates for a six-fold reduction in space [11, Sect. 1]. The optimizations below reduce that overhead by more than a factor of 100.

iv Computing the residues and arithmetic optimizations. It remains to describe how each residue $r_j = \bar{V} \bmod p_j$ is actually formed on the small ℓ-qubit register, and here Gidney [11] introduces several optimizations on top of the residue arithmetic of [10]. The naive route would compute r_j as a product of n_e controlled modular multiplications, one per exponent qubit, exactly as in the schoolbook exponentiation of Shor's algorithm. This costs $O(n_e)$ modular multiplications per prime. The first optimization, which was already proposed by the authors of [10] replaces this product by a sum of discrete logarithms. Fixing a generator g_{p_j} of $\mathbb{Z}^*_{p_j}$, each precomputed factor $M_k = g^{2^k} \bmod N$, reduced modulo p_j, can be written as $(M_k \bmod p_j) = g_{p_j}^{D_{p_j,k}}$ for a classically known discrete log $D_{p_j,k}$. This requires $M_k \not\equiv 0 \pmod{p_j}$, since 0 has no discrete logarithm. Gidney enforces it in the choice of the prime set. Before searching for P, he excludes every prime that divides one of the multipliers the circuit uses [11, Eq. (49)]. The prime set therefore depends on g and on the windowing parameters, and is chosen per shot. The construction of [10] instead handles these cases sepa-

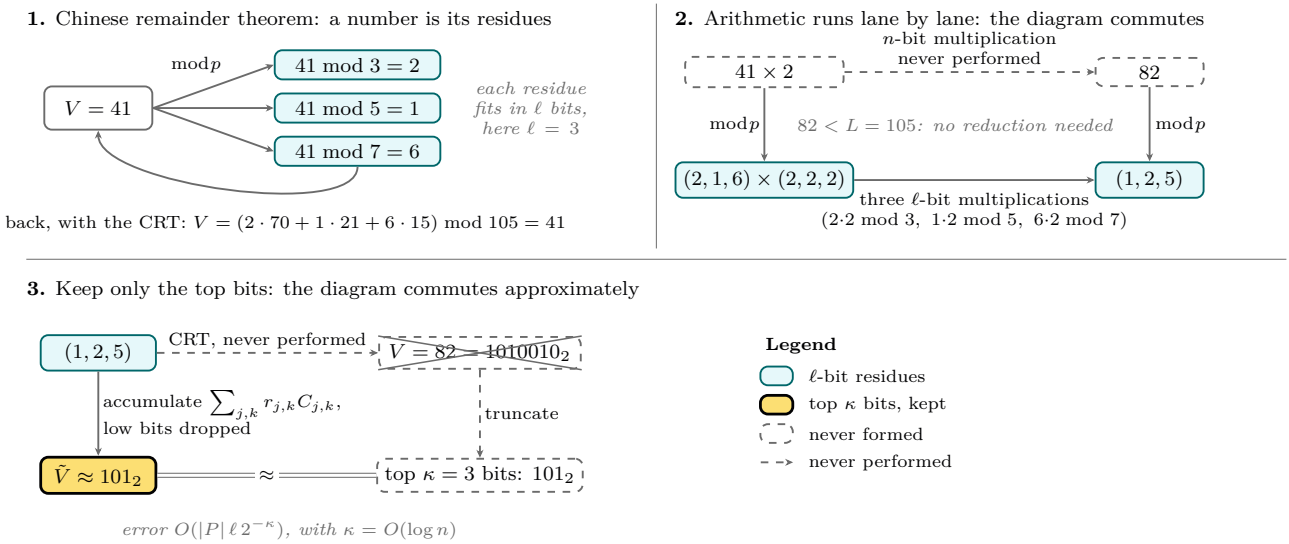


Figure 14: How the oracle stays space-efficient, on a toy residue number system with $P = \{3, 5, 7\}$ and $L = 105$. **(1)** By the Chinese remainder theorem, a number is its residues and the CRT coefficients $u = (70, 21, 15)$ rebuild it. **(2)** Arithmetic acts on each residue independently. Multiplying and then reducing modulo each prime gives the same residues as reducing first and multiplying lane by lane, so the n-bit multiplication is never performed. The product 82 stays below L, so no reduction modulo L arises. **(3)** The full value V is never assembled. The circuit accumulates the residue bits against precomputed constants and keeps only the top $\kappa = O(\log n)$ bits, which agree with the top bits of V up to the truncation error. Measuring this accumulator projects the exponent register onto the exponents consistent with the outcome, and period finding then reads r from the exponent register. The final reduction modulo N is left out of the toy. Chevignard, Fouque and Schrottenloher hash the value to its lower bits, while Gidney keeps its top bits, as shown.

rately. Under this hypothesis,

$$\begin{aligned} r_j &= \Big(\prod_{k=0}^{n_e-1} M_k^{e_k}\Big) \bmod p_j = \bar{V} \bmod p_j \\ &= g_{p_j}^{S_j \bmod (p_j-1)} \bmod p_j, \end{aligned} \tag{54}$$

with $S_j = \sum_{k=0}^{n_e-1} e_k\, D_{p_j,k}$. Here S_j is the integer sum, of at most about $\ell + \log_2 n_e$ bits. It is reduced modulo the order $p_j - 1$ of g_{p_j} only just before the exponentiation [11, Eqs. (50)–(51)]. Reducing each M_k modulo N before reducing modulo p_j is essential. It is the integer $\bar{V}$, whose residue modulo N is g^e mod N, that the CRT reconstructs, not the residue of the true power g^e, which would give an unrelated value.

Computing r_j thus reduces to accumulating the exponent S_j, a sum of at most n_e classical constants controlled on the exponent qubits, followed by a single modular exponentiation $g_{p_j}^{S_j \bmod (p_j-1)} \bmod p_j$, as depicted in Figure 15. The number of multiplications drops from $O(n_e)$ to $\operatorname{len} p_j = O(\log \log N)$.

The remaining optimizations reduce the cost of accumulating S_j. Windowing the accumulation over blocks of w_1 exponent qubits replaces individual controlled additions by table lookups. A window of w_1 qubits addresses a precomputed table holding the combined discrete-log contribution $\sum_k e_k D_{p_j,k}$ for that block. The inner loop then runs $\lceil n_e/w_1 \rceil$ times instead of n_e, at the cost of a lookup of address size w_1 per iteration. Here w_1 is a windowing parameter specific to the residue-accumulation step, independent of the windowing parameters w_e and w_l used for the main modular exponentiation. It controls the granularity of the discrete-log lookup table within each residue computation. As with the Ekerå–Håstad parameter s, increasing w_1 trades a lookup table 2^{w_1} against fewer additions, reducing gate count per residue. The residues are formed one prime at a time on a single reused register (Figure 13). As a result, the uncomputation of S_{j-1} can be merged with the computation of S_j by tabulating the differences $D'_{p_j,k} = D_{p_j,k} - D_{p_{j-1},k}$, halving the number of accumulation passes. This works because S_{j-1} and S_j are integers, so that $S_j = S_{j-1} + \sum_k e_k D'_{p_j,k}$. Reduced modulo $p_{j-1} - 1$ and $p_j - 1$, they could not be related this way, since the two moduli differ. Together these optimizations reduce the Toffoli count by more than a factor of 100 relative to [10].

A.8 Approximate Period Finding

The exact modular exponentiation $V(e) = g^e \bmod N$ satisfies $V(e + kr) = V(e)$ for every e and every integer k, since $g^r \equiv 1 \bmod N$. The approximate oracle $\hat{V}$ (see Equation (51)) computed by the residue arithmetic described above is only approximately periodic. By Equation (50), each $\hat{V}(e + kr)$ is within $\frac{\epsilon}{2}N$ of the

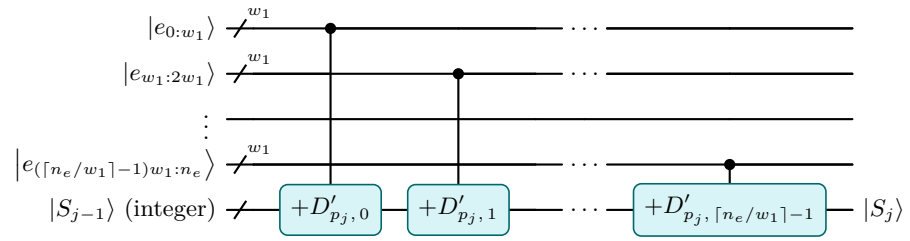


Figure 15: Accumulation of the exponent S_j for a single prime p_j. Here $S_j = \sum_k e_k D_{p_j,k}$, where $D_{p_j,k}$ is the classically precomputed discrete logarithm of $M_k \bmod p_j$ to the base g_{p_j} (Equation (54)). It is held, unreduced, on $\ell + \log n_e$ qubits. The exponent register is read in windows of w_1 qubits, indexed by k. Each gate $+D'_{p_j,k}$ adds the table entry $\sum_{i<w_1} e_{kw_1+i}\big(D_{p_j,kw_1+i} - D_{p_{j-1},kw_1+i}\big)$ selected by window k, so a single pass of $\lceil n_e/w_1 \rceil$ additions, instead of n_e, turns S_{j-1} into S_j. It merges the uncomputation of the previous prime with the computation of the current one. S_j is then reduced modulo $p_j - 1$, and a windowed modular exponentiation computes $r_j = g_{p_j}^{S_j \bmod (p_j-1)} \bmod p_j$ into the ℓ-qubit residue register of Figure 13b.

exact value $V(e+kr)$. The triangle inequality gives:

$$\begin{aligned}\Delta_N\big(\hat{V}(e+kr) - \hat{V}(e)\big) \le{}& \Delta_N\big(\hat{V}(e+kr) - V(e+kr)\big)\\ &+ \Delta_N\big(V(e+kr) - V(e)\big)\\ &+ \Delta_N\big(V(e) - \hat{V}(e)\big).\end{aligned} \tag{55}$$

The first and last terms are at most $\frac{\epsilon}{2}N$ and the middle term is 0, so the sum is at most ϵN:

$$\begin{aligned}&\Delta_N\big(\hat{V}(e+kr) - \hat{V}(e)\big) \le \epsilon\, N\\ &\text{for all } e,k \in \mathbb{Z} \text{ with } e,\ e+kr \in [0, 2^{n_e}).\end{aligned} \tag{56}$$

The algorithm measures the accumulator holding $\tilde{V}$ and then applies a quantum Fourier transform to the exponent register. The transform recovers r if, after the first measurement, the exponent register is close to a state whose amplitudes are constant on each coset $e + r\mathbb{Z}$. Measuring the full accumulator and obtaining V^* keeps the exponents e with $\tilde{V}(e) = V^*$. Because of the drift, this set contains some elements of a coset and misses others. The resulting state is not periodic, and the Fourier transform cannot recover r. There are two ways around this.

i Measure only the most significant bits. Write $M = N \gg \bar{\kappa} = \lfloor N/2^{\bar{\kappa}} \rfloor$ for the modulus of the accumulator (see Equation (52)). Since $\bar{\kappa} = \log N - \kappa$, with $\log N$ the bit length of N, it satisfies $2^{\kappa-1} \le M < 2^{\kappa}$. A natural fix is to measure only the top t bits of the accumulator. The drift changes these bits only when the value lies within $\frac{\epsilon}{2}M$ of a boundary between two of the 2^t buckets. This concerns a fraction of about $\epsilon\, 2^t$ of the cosets. Since the read-out is partial, the support of the exponent register collapses to the union of the roughly $r/2^t$ cosets whose values share the measured top bits. This resembles the hashed oracle of Section A.6, with a fixed hash instead of a random one. The unmeasured low bits, however, remain entangled with the exponent register. Tracing them out leaves the exponent register in the same mixed state as measuring them, which is the failing full read-out above. Clearing them requires running the exponentiation in reverse, which roughly doubles the dominant cost [11]. Superposition masking avoids this reverse pass [11].

ii Superposition masking. Superposition masking [60, 61, 64] reveals no more than a top-bit read-out with $2^t = 1/S$, yet it measures every qubit of the accumulator and so leaves nothing to uncompute. The key idea is to never let the measurement reveal the low bits of $\tilde{V}$. The $(\kappa+1)$-qubit accumulator that receives the truncated additions of Equation (52) is simply not initialized to $|0\rangle$. It is initialized to the uniform superposition

$$|\sigma\rangle = \frac{1}{\sqrt{W}} \sum_{s=0}^{W-1} |s\rangle\,, \qquad W = 2^b,$$

where $S = W/M \in (0,1)$ is the *fraction of the value range* $[0, M)$ covered by the mask. It is optimized below, then rounded so that W stays a power of 2. Note that S is a fraction of the values, not of the qubits. A uniform superposition over $W = 2^b$ consecutive values is prepared by one Hadamard on each of the $b = \log_2 W = \log_2(SM) \approx \kappa - \log_2(1/S)$ lowest qubits of the accumulator. Halving S removes one qubit from the superposition, and the register keeps its $\kappa + 1$ qubits whatever S is. Note that the mask is directly applied to the accumulation register. Placing it in a separate register would require an extra adder to combine it with $\tilde{V}$, and that register would remain entangled with the output.

Initializing the accumulator to $|\sigma\rangle$ only differs from the standard register initialization by those Hadamards.

Every truncated addition then acts on the masked register. Each one is a cyclic shift of the basis states modulo M, and its circuit depends only on the width of the register, not on its contents. A superposition of W consecutive values shifted by a constant stays a superposition of W consecutive values. At the end of

the accumulation procedure, after the last prime, the joint state is therefore

$$\frac{1}{\sqrt{2^{n_e}W}}\sum_e |e\rangle \otimes \sum_{s=0}^{W-1} \left|\left(s+\tilde{V}(e)\right) \bmod M\right\rangle,$$

and the whole accumulator is measured, returning some V^*. Since addition commutes, starting from $|\sigma\rangle$ is equivalent to adding a uniform s at the end.

In the units of $\hat{V}$, that is after multiplying by $2^{\bar{\kappa}}$, the mask spans $SM\,2^{\bar{\kappa}} \approx SN$ values, as presented in [11]. Physically it still lives inside the $\kappa+1$ qubits of the accumulator. A register holding SN values at full precision would need $\log_2(SN) \approx n$ qubits and defeat the whole construction. Three observations explain why masking works.

For an integer a, write

$$|B_a\rangle = \frac{1}{\sqrt{W}}\sum_{s=0}^{W-1} |(a+s) \bmod M\rangle$$

for the mask state shifted by a, so that $|\sigma\rangle = |B_0\rangle$. In this notation, the state displayed above is $\left|\tilde{\psi}\right\rangle = 2^{-n_e/2}\sum_e |e\rangle \left|B_{\tilde{V}(e)}\right\rangle$. We compare it with the ideal state $|\psi\rangle$, defined in the same way with $\tilde{V}(e)$ replaced by $V(e) \gg \bar{\kappa}$. Since V is exactly periodic, the accumulator part of $|\psi\rangle$ depends on e only through its coset $e + r\mathbb{Z}$.

Observation (1): the drift costs an infidelity of at most ϵ/S. For each e, the boxes $\left|B_{\tilde{V}(e)}\right\rangle$ and $\left|B_{V(e)\gg\bar{\kappa}}\right\rangle$ are offset by an integer η_e, the representative of $\tilde{V}(e) - (V(e) \gg \bar{\kappa})$ modulo M in $[-M/2, M/2)$. Two boxes of width $W \le M/2$ offset by η, with $|\eta| \le W$, share $W - |\eta|$ values, so

$$\langle B_a | B_{a+\eta}\rangle = 1 - \frac{|\eta|}{W}.$$

Without a mask, $W = 1$ and an offset of one unit already makes the two boxes orthogonal. This is why the full measurement fails. With the mask, condition (i) of Definition A.6 gives, after division by $2^{\bar{\kappa}}$, $|\eta_e| \le \frac{\epsilon}{2}M$ up to two units of rounding, one from the floor in $V(e) \gg \bar{\kappa}$ and one from $N \ne M\,2^{\bar{\kappa}}$. Hence $|\eta_e|/W \le \epsilon/(2S)$. The branches $|e\rangle$ are orthogonal and equally weighted, so $\langle\psi|\tilde{\psi}\rangle$ is the average of these overlaps over e. When $\epsilon \le 2S$, this gives

$$1 - \left|\langle\psi|\tilde{\psi}\rangle\right|^2 \;\le\; 1 - \left(1 - \frac{\epsilon}{2S}\right)^2 \;\le\; \frac{\epsilon}{S}.$$

The rounding units add a term of order $1/W$, which is negligible for the parameters below. Following [11], we charge this infidelity as a per-shot failure probability and continue with the ideal state $|\psi\rangle$.[10] This charge is a bookkeeping device, not a description of individual shots. For a fixed outcome V^*, a coset is cut, with some of its exponents kept and others discarded, only when V^* falls within the drift of its box edges. The outcomes that cut a given coset form at most a fraction 2ϵ of $\mathbb{Z}_M$. Each exponent is kept by W outcomes, and its coset is cut for at most $2\epsilon M$ of them. Since $\Pr[v]$ is proportional to $|K_v|$, the cut cosets hold on average at most a fraction $2\epsilon M/W = 2\epsilon/S$ of the kept exponents, without any assumption on the coset values. For the real parameters the cosets far outnumber $1/\epsilon$, so cut cosets appear in essentially every shot. This weight counts every kept exponent of a cut coset, so it bounds the affected exponents rather than the infidelity. The linear charge books this weight as if a fraction ϵ/S of the shots were lost.

Observation (2): the ideal read-out keeps about a fraction S of the exponents. Measuring the accumulator of $|\psi\rangle$ returns a value V^* and keeps the exponents whose ideal box contains it,

$$|\psi_2\rangle \;\propto\; \sum_e |e\rangle\cdot\mathbf{1}\left[\left(V^* - (V(e)\gg\bar{\kappa})\right) \bmod M \in [0, W)\right].$$

Here $\mathbf{1}[\cdot]$ equals 1 when its condition holds and 0 otherwise. The condition depends on e only through $V(e)$, so each coset of $H = r\mathbb{Z}$ is kept or discarded as a whole, and the kept set stays periodic. A coset is kept when its value lies among the W values ending at V^* modulo M, a fraction S of $\mathbb{Z}_M$. Write K_v for the set of exponents kept on outcome v, so that $\Pr[v] = |K_v|/(2^{n_e}W)$ and $\sum_{v\in\mathbb{Z}_M}|K_v| = 2^{n_e}W$. Write y for the outcome of the QFT. The post-measurement state is uniform on K_v, so $\Pr[y=0 \mid v] = |K_v|/2^{n_e}$ exactly, for any kept set. Averaging over v and applying the Cauchy–Schwarz inequality gives

$$\Pr[y=0] = \sum_{v\in\mathbb{Z}_M}\frac{|K_v|^2}{2^{2n_e}W} = S\left(1+\mathrm{CV}^2\right) \ge S,$$

where CV is the coefficient of variation of $|K_v|$ over uniform $v \in \mathbb{Z}_M$. The same computation holds for $\left|\tilde{\psi}\right\rangle$, with $\tilde{V}(e)$ in place of $V(e) \gg \bar{\kappa}$. The zero frequency therefore costs at least S. It costs about S when the values $V(e) \gg \bar{\kappa}$ are equidistributed in $\mathbb{Z}_M$ at the scale W, so that $|K_v|$ hardly depends on v. This equidistribution is the assumption behind $\delta_\perp \approx S$. The read-out resembles the hashed oracle of Section A.6, with S in the role of the collision rate 2^{-t}. The analogy is not exact. The window test is not a universal hash in the sense of [9, Def. 1], since two values collide with a probability that depends on their distance. The guarantee of [9, Thm. 7] therefore does not apply to the informative peaks. For those peaks, Gidney models the kept set as random [11, Eqs. (37)–(42)]. In this model they share the remaining weight with unchanged relative weights [11, Eq. (42)], which underlies his cost estimate [11, Assumption 2]. His numerical experiments support the model for small N but do not prove it.

[10] Charging infidelity directly as a failure probability is a heuristic. The rigorous route goes through the trace distance, $\sqrt{1-\left|\langle\psi|\tilde{\psi}\rangle\right|^2} \le \sqrt{\epsilon/S}$, which is larger. The linear charge is the one used in [11] to choose the parameters.

Observation (3): no garbage is left to uncompute. All $\kappa + 1$ qubits of the accumulator are measured, so no part of it remains entangled with the exponent register, and the exponentiation need not be run backwards. This holds for any full measurement, including the unmasked one that fails above. What the mask adds is that this full measurement is harmless. Given e, the outcome $V^* = (s + \tilde{V}(e)) \bmod M$ is uniform over W consecutive values. It therefore carries at most $\log_2(M/W) = \log_2(1/S)$ bits of information about $\tilde{V}(e)$, and essentially nothing about the low bits that hold the drift. Observations (1) and (2) quantify what this costs. Compare with the top-bit read-out for $2^t = 1/S$. Both reveal the same amount of information, and in both the drift cuts a fraction of order $\epsilon\, 2^t = \epsilon/S$ of the kept cosets in every shot. Masking differs in two ways. Its bucket boundaries sit at a random offset instead of fixed positions, and it leaves no unmeasured register behind. Figure 16 illustrates the three cases on a toy example.

iii Anatomy of the accumulator. The three observations assign a role to each of the κ value bits of the accumulator. The extra wrap-around qubit plays no role here. From the least significant bit upwards, the accumulator splits into three layers.

- The drift occupies the low $\log_2(\epsilon M)$ bits. The computed value deviates from the ideal one by at most $\frac{\epsilon}{2}M$ in either direction, a band of total width ϵM.
- The clean buffer occupies the next $\log_2(S/\epsilon)$ bits. Together with the drift band it forms the $\log_2(SM)$ bits covered by the mask. The ratio S/ϵ of the mask width to the drift width is the inverse of the corruption rate ϵ/S of observation (1).
- The top $\log_2(1/S)$ bits are what the measurement effectively reveals.

At the optimum $S = \sqrt{\epsilon}$ derived below, the buffer and the revealed part have the same width $\log_2(1/S)$. For RSA-2048 with Gidney's parameters ($\kappa = 33$, called f in [11], and $P_{\text{deviant}} \approx 1.25\,\%$, the per-shot failure rate defined in the success-rate paragraph below), this gives $S \approx 0.6\,\%$, about 7 revealed bits, a buffer of about 7 bits and a drift of about 18 bits (Figure 17).[11]

Remark A.9 (Back to two dimensions). The argument above carries over to the two-dimensional setting of Section A.4. Replace e by (a, b) and the coset $e + r\mathbb{Z}$ by a coset of the lattice H. The bound (56), the box overlap and the fact that the masked read-out keeps or discards whole cosets use only that V is constant on cosets, so they hold in any dimension. The zero-frequency identity of observation (2) holds in any dimension as well. The equidistribution behind $\delta_\perp \approx S$ remains an assumption, as in one dimension. Gidney presents his analysis for one-dimensional period finding and states that it is meant for the Ekerå–Håstad variant [11, Sect. 2]. In two dimensions, the relative weights of the informative peaks are therefore a modelling assumption as well. This two-dimensional setting is the one used by the full algorithm of Section A.9.

iv Success rate. In the language of Definition A.6, $\hat{V}$ satisfies condition (i) with the ϵ of Equation (50), and masking is the read-out R. We do not establish condition (ii) with the worst case over instances that $\delta_\perp$ and $\delta_\times$ require. The only worst-case statements are $\delta_\times \le \sqrt{\epsilon/S}$ and $\delta_\perp \ge S$, and the rates below are heuristic estimates for the instances at hand. Let $P_{\text{deviant}} = \delta_\perp + \delta_\times$ be the probability that this read-out spoils a shot. The two terms move in opposite directions as S varies. A shot aborts when the QFT returns the zero frequency, which carries no information and is discarded. By observation (2), this happens with probability $\delta_\perp \ge S$, with near equality when the coset values are equidistributed at the scale of the mask. This near equality is an assumption, not a proven bound. A shot is corrupted when the drift becomes visible. By observation (1), this happens with probability $\delta_\times \approx \epsilon/S$. This is also a heuristic, because it charges an infidelity as a probability (see the footnote to observation (1)). Counting all of this mass as corruption is conservative, since part of it may land on the zero frequency and be discarded. Adding the two rates gives

$$\begin{aligned} P_{\text{deviant}} &\lesssim S + \epsilon/S, \\ \min_S \big(S + \epsilon/S\big) &= 2\sqrt{\epsilon} \quad \text{at } S = \sqrt{\epsilon}. \end{aligned} \tag{57}$$

At this optimum the abort and corruption rates are equal, and $P_{\text{deviant}} \lesssim 2\sqrt{\epsilon}$. With the rigorous charge $\delta_\times \le \sqrt{\epsilon/S}$ instead, and still $\delta_\perp \approx S$, the optimum moves to $S = (\epsilon/4)^{1/3}$. It then gives $P_{\text{deviant}} \lesssim 3(\epsilon/4)^{1/3} \approx 1.9\,\epsilon^{1/3}$.

Equation (57) also fixes the truncation length κ. Write the drift of Equation (50) as $\epsilon = c\, A_{\text{acc}}\, 2^{-\kappa}$, where A_{acc} is the number of truncated additions into the accumulator and one truncated addition shifts the value by at most $\frac{c}{2}\, 2^{-\kappa} N$. Setting $2\sqrt{\epsilon} = P_{\text{deviant}}$ and solving for κ gives

$$\kappa \;=\; \underbrace{\log_2(c A_{\text{acc}})}_{\text{drift}} \;+\; \underbrace{2\log_2\!\big(2/P_{\text{deviant}}\big)}_{\text{buffer+revealed}}. \tag{58}$$

The first term pays for the drift. Without windowing, $A_{\text{acc}} = |P|\,\ell = \Theta(n_e n)$, so this term is $\log_2(n_e n) + O(1) = O(\log\log N)$. The second term

[11] Back-of-envelope values, taking $\kappa = 33$ and P_{deviant} from the $n = 2048$ row of Table 5 of [11] as inputs. We take this failure rate as given and do not recompute it from the circuit of [11]. The values above are those that our bound (57) needs to reach it, obtained by inverting $P_{\text{deviant}} = 2\sqrt{\epsilon}$. The actual mask width W is a power of two, so S is close to $\sqrt{\epsilon}$ but not exactly equal to it.

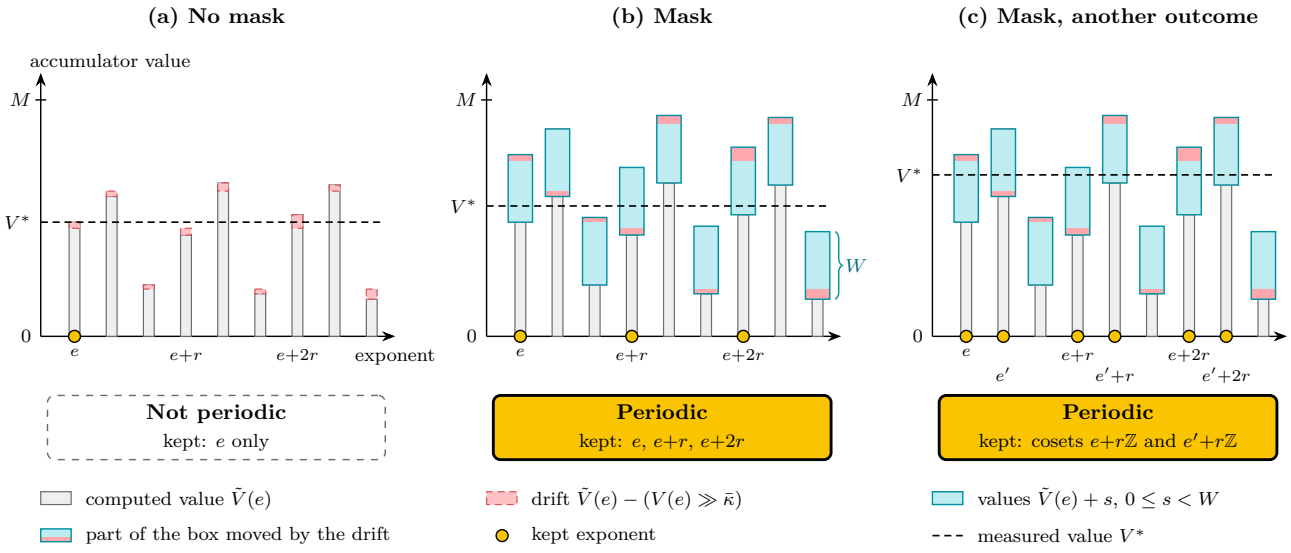


Figure 16: How superposition masking restores periodicity. Bars are the computed values $\tilde{V}(e)$ of the accumulator. The exact value $V(e) \gg \bar{\kappa}$, in the same units, is constant on each coset $e + r\mathbb{Z}$, and $\tilde{V}(e)$ is not. **(a)** Without mask, only the exponents with $\tilde{V}(e) = V^*$ are kept. The drift separates e from $e + r$ and $e + 2r$, so the kept set is not periodic. **(b)** With the mask, when the input register holds e, the accumulator holds the $W = SM$ consecutive values $\tilde{V}(e) + s$, $0 \le s < W$ (teal box). Without drift, all boxes of a coset would start at the same value $V(e) \gg \bar{\kappa}$. The drift shifts each box, and the salmon strip is the part of the box shifted out of that common position. An outcome within the drift of the box edges keeps only part of the coset. For the real parameters, every outcome cuts some cosets in this way, and they hold at most about a fraction $2\epsilon/S$ of the kept exponents (observation (1)). **(c)** Same state as in (b), another outcome. The boxes of the coset of e' also contain V^*, so this coset is kept too. The kept set is a union of cosets and still has period r. For the real parameters, every outcome keeps about a fraction S of all cosets whole, as here, and cuts a fraction at most about 2ϵ of them as in (b). The zero frequency of the QFT then has weight about S (observation (2)).

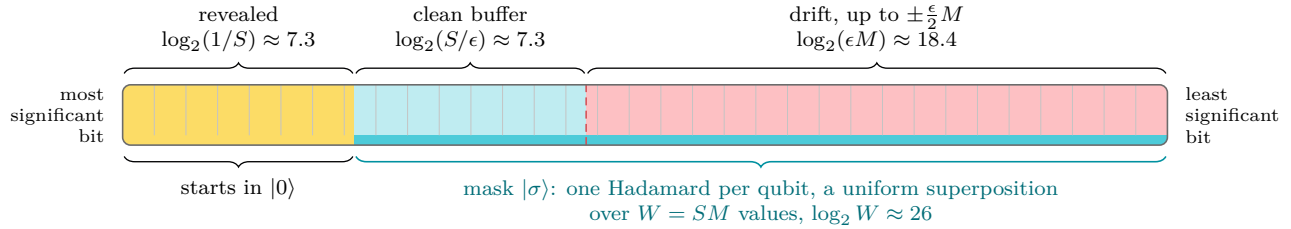


Figure 17: Bit layout of the $\kappa = 33$-bit masked accumulator for RSA-2048, with the most significant bit on the left (one extra wrap-around qubit not shown). It shows, bit by bit, the accumulator of Figure 13b. All qubits are measured at the end. Because of the mask, the measured value is uniform over W values for every exponent, so the measurement reveals at most $\log_2(1/S)$ bits of information about $\tilde{V}$. The clean buffer between those bits and the drift sets the corruption rate: its width $\log_2(S/\epsilon)$ gives the heuristic rate ϵ/S. Widths are in bits, derived from the failure rate of [11, Table 5] with $S = \sqrt{\epsilon}$ and $M = 2^\kappa$, before W is rounded to a power of two.

equals $\log_2(1/\epsilon)$, the combined width of the buffer and the revealed bits. It depends only on the target failure rate and grows by two bits each time P_{deviant} is halved. For RSA-2048 at $P_{\text{deviant}} = 1.25\,\%$, it is about 14.6 bits. Gidney's parameter search gives $\kappa = 33$ for this target [11], so his windowed circuit spends the remaining ≈ 18 bits on drift. We infer this value from $\kappa = 33$ and do not derive it from the windowing parameters. Under the rigorous charge, the second term becomes $3\log_2(3/P_{\text{deviant}}) - 2 \approx 21.7$ bits at the same target, and κ would grow to about 40.

A.9 Assembling Modern Shor

The previous subsections built the modern algorithm one relaxation at a time. This subsection makes one choice per stage, assembles the choices into a single pipeline, and instantiates it for an $n = 2048$-bit RSA modulus. The pipeline uses the short-discrete-log reduction, the Ekerå–Håstad register shape with Ekerå's graded post-processing [52], and the residue-arithmetic oracle with masking. The $(\kappa+1)$-qubit accumulator of that oracle already compresses the output, so the hash of Section A.6 is not used. Table 3 marks these choices. We follow the run from the choice of g to the recovered factors p and q, and we

state, at each stage, the failure probability already derived above. Composing them gives one number: the *algorithmic error rate* of the full pipeline. This rate is algorithmic. It comes from the structure of the algorithm itself, from the randomness of g, from the statistics of the measured frequency pairs, and from the noise budget of the space-efficient oracle. Two of its inputs are heuristic: the masking rates $\delta_\perp \approx S$ and $\delta_\times \approx \epsilon/S$ (Section A.7), and the ≈ 1 % failure floor of Ekerå's post-processing, which is measured by simulation [52]. It says nothing about the physical qubits that execute the algorithm. A real run must also survive the physical error rate of the hardware, which is the subject of a separate part of this work. Table 4 summarizes the failure budget derived below, before we walk through each row in turn.

At this level of detail we keep visible only the parameters that set the algorithmic error rate: the lattice parameter s of Section A.5, the truncation length κ, and the mask width S of Section A.7. The windowing parameters w_e and w_l, the lookup table sizes, and the exact set of RNS primes P mainly set the gate count and the qubit count of the resulting circuit. They are not neutral for the success probability, however. Windowing changes the number A of truncated additions, and the prime set fixes $\Delta_N(L)$. At fixed κ, both change the drift ϵ through Equations (49) and (58), and hence P_{deviant}. We take them as fixed by Gidney's parameter search, which sets $\kappa = 33$ for the resulting drift. Their concrete values are given in Appendix B and used for the resource estimate in Appendix C.

i Classical pre-processing. Fix the 2048-bit RSA modulus $N = pq$. Draw a uniformly random $g \in \{2, \ldots, N-2\}$ and compute $\gcd(g, N)$ with the Euclidean algorithm. If the gcd is not 1, it is p or q, and the factorization is already found. This early success happens with probability $O(2^{-1024})$ for 1024-bit primes, so it plays no role in the budget, and we assume $\gcd(g, N) = 1$ from here on. Compute classically the value

$$x = g^{(N-1)/2} \bmod N. \tag{59}$$

By the reduction of Section A.4, $x = g^d \bmod N$ for the short logarithm $d = (p + q - 2)/2$, of bit-length $m \approx n/2 = 1024$. Recovering p and q is now equivalent to recovering d. This step is deterministic. It introduces no failure probability of its own.

ii Fixing the register-shape parameters for $n = 2048$. One of the three register-shape parameters is a genuine trade-off and we keep it symbolic. Recall from Section A.5 that the lattice parameter $s \geq 2$ sets the exponent register length $n_e = \left(\frac{1}{2} + \frac{1}{s}\right) n$. For $n = 2048$ this ranges from $n_e = 2048$ at $s = 2$ down toward 1024 for large s, trading qubits for shots. The other two are not free in the same sense. Once a target per-shot failure rate P_{deviant} is chosen, the mask width is fixed at its optimum $S = \sqrt{\epsilon}$ by Equation (57), and the truncation length κ follows from Equation (58). That equation splits κ into two parts. The drift part, $\log_2(cA)$, counts the truncated additions into the accumulator. Without windowing it is $\log_2(n_e n) \approx 21$ to 22 for $n = 2048$, and windowing lowers it. The precision part, $2\log_2(2/P_{\text{deviant}})$, buys the clean buffer and the revealed bits of Figure 17. It is not a hidden constant: it grows by two bits each time the target failure rate is halved. Gidney's optimized parameter search [11] for $n = 2048$ (with $s = 8$, so $n_e = 1280$) lands on $\kappa = 33$ bits for a mask-and-truncation failure rate $P_{\text{deviant}} \approx 1.25$ %. We take $\kappa = 33$ as an input. Of it, $2\log_2(160) \approx 14.6$ bits are the precision part, so the windowed drift term accounts for the remaining ≈ 18 bits, down from 21 to 22 without windowing.

iii One shot of the quantum core. With g, the target x, and (s, κ, S) fixed, the algorithm runs the following quantum core, and repeats it.

1. *State preparation.* Two registers of size $\tau_s + m$ and τ_s, with $\tau_s \approx m/s$, are put in a uniform superposition by Hadamards.

2. *Oracle evaluation.* The residue-arithmetic oracle of Section A.7 computes the top κ bits $\tilde{V}$ of $g^a x^{-b} \bmod N$, so that $\hat{V} \approx g^a x^{-b} \bmod N$, and adds them into the $(\kappa{+}1)$-qubit accumulator, initialized to a mask of width $\lceil SM \rceil$. The whole accumulator is then measured.

3. *Fourier transform and measurement.* A semi-classical QFT on the two exponent registers returns a pair (j, k).

Whether a shot ends up contributing to the recovery of d depends on how (j, k) relates to the hidden logarithm, and on how much noise the oracle introduced. Ekerå and Håstad's original analysis of the first dependency singles out a *good* pair, one meeting the constructive-interference condition of Equation (38), and shows that this happens with probability at least $1/8$. Good pairs cannot be recognized one at a time, so their read-out runs about $8s$ shots and tries every subset of s pairs against the check $x = g^d$. Ekerå's later post-processing [52] refines this. Rather than a hard good/bad split, it grades every possible outcome and combines the graded evidence directly in the lattice step below, so that a pair the simple analysis would count as bad can still contribute. Under this read-out a single pair asserts nothing that can be false. A bad pair is therefore neither an abort nor a corruption, and its cost moves into the success probability q_{pp} of the lattice step (Lemma A.10). This is the post-processing this section uses from here on, and it needs on average only about $s + 1$ shots, rather than the roughly $8s$ of the subset search.

Stage	Failure	Rate
Pick g (once per run)	$\gcd(g, N) \neq 1$: a factor found directly (early success)	$O(2^{-1024})$
RNS oracle + masking (per shot)	mask abort	$\approx S$
RNS oracle + masking (per shot)	corruption	$\approx \epsilon/S$
Lattice recovery of d (once, from enough consistent shots)	post-processing fails despite enough consistent shots [52]	≈ 1 %
Check $q(2d + 2 - q) = N$ (once per attempt)	(catches remaining corruption)	—

Table 4: Failure budget of the RSA-2048 pipeline, one row per stage. Shot counts throughout this section already use Ekerå's graded post-processing [52] (see below), not a hard good/bad filter on the measured frequency pair. The last row is not a failure. It is the classical check that turns every remaining corruption into a caught and retried attempt rather than a wrong output.

Given Ekerå's post-processing, a shot can still be wasted or corrupted, but only through the oracle's own noise, from Section A.7. The masked measurement can abort on its own, with probability about S, when the shot collapses onto the uninformative zero-frequency outcome. It can also corrupt, with probability about ϵ/S (a heuristic estimate, Section A.7), returning a constraint built from the wrong value of $\tilde{V}$ without any visible sign that it is wrong. Together these two contributions give $P_{\text{deviant}} \lesssim S+\epsilon/S$, which the optimal choice $S = \sqrt{\epsilon}$ brings down to $2\sqrt{\epsilon}$, as in Equation (57). Figure 18 traces this per-shot budget: every shot reaches Ekerå's post-processing unless the oracle's own noise wastes or corrupts it first.

iv Classical post-processing. Shots are collected until Ekerå's graded post-processing [52] can support a single consistent value of d, which for RSA-2048 needs on average close to $s+1$ shots (made precise in the algorithmic error rate paragraph below). The lattice-based recovery of Section A.5 then returns a candidate d: a bounded-distance decoding search in a lattice of small, constant dimension, which succeeds with probability around 99 % once given enough consistent shots [52]. The candidate d gives $p+q = 2d+2$, and p, q are the two roots of $q(2d + 2 - q) = N$. This last equation, together with the condition $1 < q < N$, is also the check that the whole run succeeded. It holds if and only if d was correct. Its roots q and $2d+2-q$ are then integers whose product is N. Since $N = pq$ with p and q prime, the only factorizations of N into two positive integers are $\{p, q\}$ and $\{1, N\}$, and the condition $1 < q < N$ excludes the second. The roots are therefore p and q, so $2d + 2 = p + q$, which fixes d. The excluded case would need $d = (N-1)/2$, which the shortness of d rules out anyway.

This last point deserves emphasis. A corrupted shot does not make the algorithm output a wrong factorization. It makes the lattice step fail its own consistency check, since a corrupted constraint is inconsistent with the other good ones. Corruption costs a subset search or a whole attempt rather than a single shot (Lemma A.10), but like an abort it never produces a wrong and undetected answer. Factoring is self-verifying. p and q are only ever accepted once $pq = N$ has been checked. In practice, a run with too many corrupted shots is not simply discarded either [11]. One can first retry the lattice step over different subsets of the collected shots. Only if that also fails does one restart with a fresh g. For the person running the algorithm, every failure mode in this section is the same kind of event. It wastes an attempt or a subset, which is then retried. It never yields a silently wrong RSA key. Figure 19 places this retry loop around the full pipeline, from g to p, q.

v Expected number of shots. The algorithm does not fix a shot count in advance. It draws shots one at a time. It stops as soon as Ekerå's post-processing recovers a d that passes the check $q(2d + 2 - q) = N$. Let R be the number of shots this takes in one attempt, that is, with a single fixed g. This is a negative-binomial quantity, the number of draws needed to accumulate the $s+1$ consistent shots the lattice step requires. Ekerå's post-processing already absorbs the frequency-pair statistics, so the per-shot bookkeeping reduces to the RNS oracle's own noise. A fraction $P_{\text{deviant}} \leq 2\sqrt{\epsilon}$ of shots is wasted or corrupted by the masked, approximate oracle (Equation (57)), which inflates the count by $1/(1-P_{\text{deviant}})$. A further factor 0.99 covers the residual chance that the post-processing fails even on a clean sample [52].

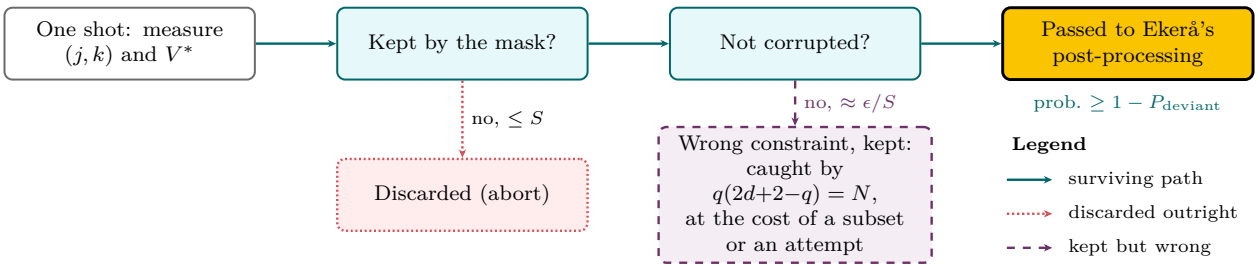


Figure 18: Per-shot failure budget contributed by the RNS oracle and its masked read-out. The teal path survives and is handed to Ekerå's graded post-processing [52]. On the salmon path the shot is discarded outright. On the dashed purple path the shot is kept but wrong. This corruption is not detected here but is caught downstream by the factorization check, at the cost of another subset of shots or a fresh attempt (Lemma A.10).

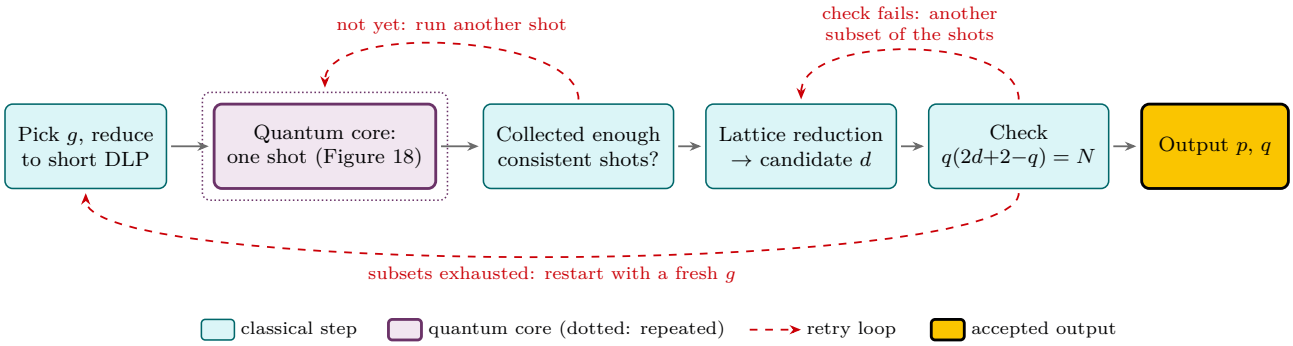


Figure 19: The full RSA-2048 pipeline. Teal: classical steps. Purple: the quantum core, repeated until enough consistent shots, in Ekerå's sense [52], are collected. Salmon, dashed: the three retry loops, for an insufficient shot count, for a candidate d that fails the final check (retry the lattice step on another subset of the shots), and for exhausted subsets (restart with a fresh g). Yellow: the accepted output, reached only once $pq = N$ has been verified.

Collecting the $s + 1$ shots the lattice step needs then costs, on average,

$$\mathbb{E}[R] \approx \frac{s+1}{(1 - P_{\text{deviant}}) \cdot 0.99}. \tag{60}$$

For RSA-2048, with $s = 8$ and $\kappa = 33$ and $P_{\text{deviant}} \approx 1.25\,\%$, both taken from [11], Equation (60) gives $\mathbb{E}[R] \approx 9.2$ [11]. The baseline is the $s + 1$ shots, and the RNS and masking noise is a percent-level correction on top. The value $s + 1$ is itself the asymptotic count, and the finite-n figures we quote are read from Ekerå's tables [52].

Equation (60) is the estimate of Gidney [11]. It rests on his Assumption 2 and charges a corrupted shot like an aborted one, as a single lost shot. A corruption can in fact cost a whole attempt. Lemma A.10 below charges it that way and gives a heuristic worst case of ≈ 10 shots.

This is also where the choice of masking over hashing, made when we fixed the pipeline earlier, pays off concretely. With a t-bit hash, the output compression of Section A.6 sends a fraction 2^{-t} of the informative shots to the zero outcome, a factor $1/(1 - 2^{-t})$ in shots. The penalty shrinks as t grows, but only by spending output qubits. Masking instead reaches $P_{\text{deviant}} \approx 1.25\,\%$ inside the $(\kappa{+}1)$-qubit accumulator it needs anyway. The implementation of [10] uses the 1-bit extreme, where zeros appear in half of all runs.

For RSA-2048 it takes $s = 17$ and $\mu = 20$ measurement results, following Ekerå's tables, and therefore needs $2\mu = 40$ runs on average, against the 9.2 shots of Equation (60) at $s = 8$. The two counts differ in s as well as in the read-out, so the ratio is not due to masking alone.

vi A Las Vegas procedure. The check $q(2d + 2 - q) = N$ makes the whole pipeline self-verifying. A run never returns a wrong (p, q). A corrupted shot, or a batch that fails post-processing, makes the check fail, and the algorithm responds by retrying the lattice step over subsets of the collected shots, or by restarting with a fresh g [11]. The algorithm is thus Las Vegas. Its only failure mode is exhausting a chosen shot or restart budget before the check passes. We define P_{alg} as the probability of that event. Independent fresh-g attempts each succeed with a constant probability p_{ok} close to 1, so after k such restarts,

$$P_{\text{alg}} \leq (1 - p_{\text{ok}})^k,$$

with each restart costing about $\mathbb{E}[R]$ shots. Lemma A.10 below bounds $1 - p_{\text{ok}}$ by about 7 % for RSA-2048. This is a heuristic worst case that ignores subset retry, which can only lower it. A budget of a few times $\mathbb{E}[R]$ therefore drives P_{alg} far below any target. At $R = \mathbb{E}[R]$ the algorithm sits near the median

of its own shot distribution, where failure is of order one half, so smallness comes only from running a constant factor past the expected count, never from the expected count itself. One modeling caveat applies to both equations. They treat the per-shot deviant rate as independent across shots. It is not exactly independent. Within a single run g is fixed, so the mask-abort rate carries a common, g-dependent skew [11], and the corruption part is a coherent error bounded through trace distance rather than a classical coin [10]. This correlation affects the spread of the shot count, not its mean or the Las Vegas guarantee. Both [11] and [10] validate the mean directly by simulation.

Lemma A.10 makes p_{ok} precise and states how aborts and corruptions combine across shots, attempts and restarts.

Lemma A.10 (Aborts and corruptions across shots and attempts)**.** *Fix one attempt, that is, one value of g. Assume that each shot aborts with probability at most $\delta_\perp$ and corrupts with probability at most $\delta_\times$, conditionally on all earlier shots. Assume that the lattice step consumes $s+1$ kept shots and, when none of them is corrupted, returns the correct d with probability at least q_{pp}. This last assumption is heuristic. It rests on the simulations of Ekerå [52]. Then the following hold.*

1. *The attempt outputs either the correct pair (p, q) or nothing. It never outputs a wrong factorization.*
2. *(Heuristic.) The attempt succeeds with probability $p_{\text{ok}} \geq q_{\text{pp}} - (s+1)\,\delta_\times/(1-\delta_\perp)$.*
3. *If aborts are independent across shots, the attempt draws on average $(s+1)/(1-\delta_\perp)$ shots.*
4. *If restarts use independent fresh values of g, then after k attempts $P_{\text{alg}} \leq (1-p_{\text{ok}})^k$. The expected number of shots over the whole run is at most $(s+1)/\big((1-\delta_\perp)\,p_{\text{ok}}\big)$.*

Proof sketch. Item 1 holds because the check $q(2d+2-q) = N$ with $1 < q < N$ is satisfied only by the correct d. For item 2, a kept shot is corrupted with probability at most $\delta_\times/(1-\delta_\perp)$ given the earlier shots. A union bound over the $s+1$ kept shots and over the failure of the lattice step on clean input gives the claim. Item 3 is the mean of a negative binomial count. Item 4 follows from the independence of the attempts and from Wald's identity.

For RSA-2048, the optimum $S = \sqrt{\epsilon}$ splits $P_{\text{deviant}} \approx 1.25$ % into $\delta_\perp \approx \delta_\times \approx 0.63$ %. This uses the heuristic charge $\delta_\times \approx \epsilon/S$ of Section A.8. With $s+1 = 9$ and $q_{\text{pp}} = 0.99$, Lemma A.10 then gives $p_{\text{ok}} \geq 0.93$ and at most about 9.7 expected shots. The rigorous charge is the trace distance, $\delta_\times \leq \sqrt{\epsilon/S} \approx 7.9$ % at the same parameters. It gives only $p_{\text{ok}} \geq 0.27$ and at most about 33 expected shots, before subset retry (Remark A.11).

Remark A.11 (Subset retry). Gidney [11] does not end an attempt at the first failed check. He takes a few extra shots and runs the post-processing on all $\binom{t}{s+1}$ subsets of the t shots collected so far. He restarts with a fresh g only when t grows too large for this search. A single corruption therefore no longer ends the attempt. Lemma A.10 ignores this recovery, so its bound on the expected shot count is a worst case.

vii Composition with the physical error rate. P_{alg} is internal to the algorithm's randomness. It does not depend on the hardware. Running the pipeline on a physical device adds a second error source. That source enters in the same way, not as a new term. A logical error during a shot is also silent. It too surfaces only as a post-processing or check failure [11]. It is therefore caught and retried by the very mechanism above, so it inflates the expected shot count rather than creating a wrong-answer probability. The full cost of a real run is then captured by two multipliers on the $s+1$ baseline. The first is the algorithmic $1/(1-P_{\text{deviant}})$ of Equation (60). The second is a physical factor set by the per-shot logical error rate, treated in the logical error budget of Section C.7.2.ii. P_{alg} is the part of that budget that no amount of qubit quality can remove. It comes from the structure of the algorithm, not from the device that runs it.

A.10 FFDLP Algorithm

The finite field discrete logarithm problem (FFDLP) is a discrete logarithm problem in a cyclic subgroup of $\mathbb{F}_p^*$. In safe-prime groups like those of [65, 66], the subgroup has prime order $r = (p-1)/2$. The logarithm itself, however, is usually much shorter than p. This is an implementation choice, not a property of the group. Standards size the secret exponent to the target security level, for instance about 220 bits for the 2048-bit group of [66]. The hardness of the problem is therefore parametrized *both* by the bit-size n of p and by the bit-size m of the discrete logarithm.

This is an important difference with RSA instances, which are reduced to an $n/2$-bit discrete logarithm in an n-bit group. Here $m \ll n$ is possible. The exponent only has to resist generic attacks on a logarithm known to lie in an interval of size 2^m, such as Pollard's kangaroo method or baby-step giant-step restricted to that interval, which take $O(2^{m/2})$ group operations [67, 68]. The (generalized) Number Field Sieve in $\mathbb{F}_p^*$ has subexponential complexity in n and does not depend on m.

This parametrization modifies the asymptotic and practical complexities of the RNS-based truncated exponentiation. This is because both the number of RNS residues (of order mn, up to logarithmic factors) and the time to compute such a residue (of order m, up to logarithmic factors) depend on m, and have been reduced with respect to a factoring instance of

comparable security level.

Apart from these different sizes, the circuit for a single shot is unchanged.

A.11 Parameter Selection for RSA and FFDLP

While the logical qubit count is mostly determined by the input register sizes, and interacts with the post-processing parameters (such as the number of shots in a run), the logical gate count is mostly determined by the parameters of the problem: the bit-size of N (or p) and the bit-size of the discrete logarithm determine the number of RNS primes and their maximal size. It also depends on the windowing parameter in the exponent register: the larger the window size, the smaller the values in the integer domain.

For several choices of bit-size, exponent size and window size, we compute an upper bound on the total gate count. We do the following approximations:

- The RNS depends on the ambient group, the exponent size, and the precomputed windows (as we have to avoid any prime which divides one of the integers in the windows). We take a random prime for the ambient group (as this does not change the sizes), but compute the windows genuinely. We take primes of minimal bit-size 12. The input register size is computed as $m + 2\lceil m/s \rceil$, and padded to a multiple of the window size.

- The truncation length κ is taken from [11, Table 5] for the RSA rows. The FFDLP rows have fewer RNS primes, hence less drift, and we use smaller values there. We report the resulting failure rate P_{deviant} in Table 5. For the FFDLP rows it is estimated with Equation (58), scaling the drift term $\log_2(cA)$ of the RSA-2048 row in proportion to $A = |P|\ell$. This estimate is rough. Applied to RSA-1024 at $\kappa = 28$ it gives about 3.6%, against the 2.87% reported in [11]. The value of κ does not change the CCX count of the residue circuits reported in the table, but it sets the width of the $|P|\ell$ additions into the accumulator and the qubit count.

- The CCX gate count and qubit count of a residue computation circuit increase with the RNS prime bit-size. We just take the largest primes in the RNS to upper bound the cost, and multiply by the number of primes in the RNS. Note that a residue computation circuit actually merges the uncomputation of a residue with the computation of the next one.

- The window size can be freely chosen to minimize the cost. We try window sizes between 6 and 8, and observe variations of a factor at most $2^{0.2}$ in the gate count; in the end a window size 6 seems most appropriate, except for small instances.

The overall instance parameters for different instances of RSA and FFDLP are depicted in Table 5. Note that the residue circuit is computed explicitly using our Qarton code.

A.12 The Elliptic-Curve Instance

Section A.1 showed that `ECDLP` is formally the same hidden subgroup problem as `FFDLP`. The group is $G = \mathbb{Z}_r^2$, the subgroup is $H = \langle (d, 1) \rangle$, and the function is $f(a, b) = [a]P + [b](-Q)$, evaluated using elliptic curve point additions. Throughout this section n denotes the bit-length of the curve's field or order, the role the RSA modulus bit-length played in the sections above. The two are different numbers even at comparable classical security (a 256-bit curve against a 3072-bit RSA modulus), so comparing figures across the two halves should not carry over a specific value of n. In the factoring case, the literature seems to have converged to a single best algorithm, which is the one presented in the preceding sections. On the contrary, the elliptic-curve case has multiple competing trade-offs between qubit and gate count, which we detail in this section. We start by making explicit some differences with respect to the discrete logarithm instances previously seen in factoring and `FFDLP`.

The order is known and prime. For the NIST and SECG curves the (prime) order r of the base point is a published parameter, so the exponent registers can be sized to about $\log_2 r$ bits from the start. They remain power-of-two registers, and the Fourier transform on them is semi-classical and approximate, as described below. Two simplifications follow. There is no analogue of the over-provisioned register that the unknown order forced on order finding in Section A.3, where it required $2^{2n} \geq N^2$.

The logarithm is general, not short. The Ekerå–Håstad analysis [8] needs the unknown logarithm to be *short* for its lattice recovery to succeed. Concretely, in the notation of Section A.5, this is hypothesis (34), $r \geq 2^{\tau_s+m} + 2^{\tau_s} d$. Ekerå gives the sharp form of this condition, with $2^{\tau_s} - 1$ in place of 2^{τ_s} [69, Sect. 1.4]. With $\tau \approx m/s$ it allows d up to about $r^{s/(s+1)}$, that is $d \lesssim \sqrt{r}$ for $s = 1$. For factoring, the reduction of Section A.4 arranged exactly such a short logarithm. Here d is a full-length exponent, uniform on $[0, r)$. That reduction therefore does not apply, and its heuristic variant degrades as well [70]. The input-register saving is still available, but through a different route, Ekerå's algorithm for *general* logarithms, which trades runs against register size rather than exploiting shortness. We explore this saving in more details hereafter.

The group law does not reduce to multiplication. Evaluating f means adding points on the curve. The ordinary *affine* addition law represents a point by its coordinates (x, y), and it contains a division in $\mathbb{F}_q$. For $P_1 \neq P_2$ the slope is $\lambda = (y_2 - y_1)/(x_2 - x_1)$,

Instance	n	d	κ	s	Window bit-size	Integer	RNS primes	Prime bits		Dlog acc. bit-size
								Min	Max	
FFDLP-512	512	80	26	7	7	7680	591	14	14	18
FFDLP-1024	1024	160	26	7	7	30720	2012	16	16	21
FFDLP-1536	1536	180	28	7	6	59904	3974	15	16	22
FFDLP-2048	2048	224	28	7	6	100352	6291	16	17	23
RSA-1024	1024	512	28	8	6	109568	6330	18	18	28
RSA-2048	2048	1024	33	8	6	438272	22658	20	20	31

Table 5: Instance parameters for RSA and FFDLP. n is the bit-size of the ambient group (RSA key size or ambient prime for FFDLP), d is the bit-size of the discrete logarithm, κ is the number of bits of the accumulator register (named f in [11]), s determines the input register size; "integer bit-size" is the bit-size of integers represented by the RNS; "RNS primes" is the number of primes in the RNS; "min prime bits" and "max prime bits" are the minimal and maximal bit-size of primes in the RNS (we start from a given bit-size, and take all possible smallest primes). The window bit-size of the primes are selected by minimizing the $\mathrm{CC}X$ gate count. First the window size, which has more impact, then the bit-size of primes. Both have only an impact on the first decimal of the $\log_2$, and the exact implementation of the residue circuit matters here.

giving $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = \lambda(x_1 - x_3) - y_1$. Division becomes the most expensive sub-circuit, and the primary target for trade-offs. The alternative is to use *projective coordinates*, in which addition formulas do not contain divisions. These are only used in [14].

We will therefore refer in the following to two families. The affine family keeps the point in (x, y) coordinates and computes inversions. Its two members, reaching the current best trade-offs, use two different reversible implementations of the extended Euclidean algorithm (EEA). The projective family has a single member [14], which avoids inversions and reuses some of the structure of the factoring case: the output-register compression of Section A.6 with the residue arithmetic of Section A.7. While it was the most space-efficient at the time of publication, it is now beaten by [13] in both space and gate count, as we will detail in the rest of this section. A fourth resource, circuit *depth* (the number of sequential steps needed once independent gates are parallelized, rather than their raw total), is pushed by a separate line of work again. We set it aside until the comparison at the end. Figure 20 maps the three space/gate directions and the depth axis.

Figure 20 maps the resource trade-off. Table 6 maps the same three directions onto the (ϵ, δ) axes of Definition A.6, alongside the factoring relaxations of Table 3. As in that table, these three rows are alternatives, not a pipeline. Two of the three rows never abort. They either succeed exactly or corrupt. The third compresses the output to a single Legendre bit, which is the May–Schlieper move of Section A.6 with $t = 1$. About half of its runs therefore land on the discarded zero outcome, which is why its run count doubles. By the identity of Section A.8, this abort rate is at least 1/2, with near equality when the hash is balanced.

i Recycling the input register. This is not a trick specific to elliptic curves: it is the same semi-classical-QFT recycling already used for the factoring baseline of Section A.3, applied here to two registers instead of one. The two exponent registers, holding a and b, never exist in full at any one time. The w qubits of one window of the exponents enter the circuit together, as a single control block. They control one windowed point addition and are then measured, before the next window is prepared. Because the quantum Fourier transform at read-out can be performed *semi-classically* [53], each measurement outcome is a classical value that only sets the phase of the gates still to come. Nothing therefore needs to be kept coherent once it has been read. A measured window of control qubits can thus be reset and reused for the next window [54–56]. Carried through, a single recycled w-qubit window does the work of both exponent registers, saving the $\approx 2n$ qubits they would otherwise occupy [13]. This is why the (a, b) registers cost essentially nothing beyond the single w-qubit window already totaled elsewhere, and why the space figures of the recycling constructions count only the *point* registers, the workspace of the arithmetic on the curve [6, 12, 13, 62, 71–73]. The compressed construction of [14] does not recycle (Remark A.12), so its figure includes the index registers.

The input-register optimization of Section A.5 therefore buys no space on the elliptic curve. What it can still buy is fewer point additions per run, which lowers both the gate count and the depth. For factoring, the picture depends on the oracle. With recycling, the Ekerå–Håstad reduction of Section A.5 likewise buys fewer modular multiplications per run, not space. With the residue-arithmetic oracle, which cannot recycle, it saves qubits, and this is how the modern factoring pipeline uses it. Ekerå's algorithm for general logarithms with tradeoffs [74] realizes this. With a tradeoff factor $s \geq 1$ and $\ell = \lceil m/s \rceil$, its two exponent registers hold $m + \ell$ and ℓ bits, so it evaluates about $(1 + 2/s)m$ point additions per run instead of Shor's $2m$, fewer as soon as $s > 2$. The price is recovering d from several runs by lattice re-

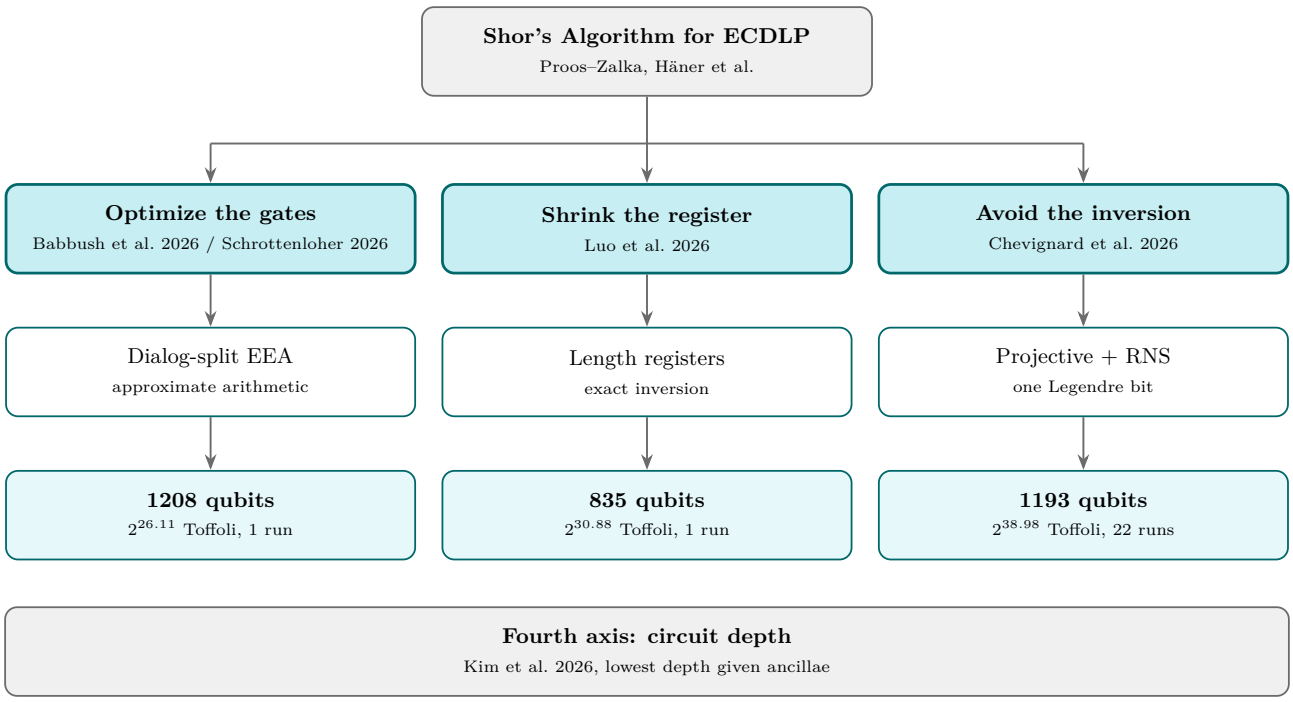


Figure 20: The elliptic-curve frontier, oriented by the resource each direction pushes: *optimize the gates*, *shrink the register*, *avoid the inversion*, and a fourth axis, circuit depth. Unlike the factoring pipeline, no algorithm dominate for all metrics. The figures are current $n = 256$ estimates.

Direction	ϵ	$\delta = \delta_\perp + \delta_\times$		**Read-out** R
		$\delta_\perp$ *abort*	$\delta_\times$ *corrupt*	
Optimize the gates	0	0	$\lesssim 2^{-6.9}$	measure the accumulated point (rare circuit failure corrupts silently, empirical rate, 95% bound)
Shrink the register	0	0	$\approx 2^{-w}$	exact arithmetic. Corrupts only when the first window is 0
Avoid the inversion	0	$\approx 1/2$	$\lesssim 0.01$	measure 1 Legendre bit, a 1-bit hash (about half the runs abort). Exact except on a sparse set of inputs, where it corrupts

Table 6: The three ECDLP directions of this section, placed on the same (ϵ, δ) axes as Table 3. All three oracles are exact ($\epsilon = 0$). The first and last rows fail on a sparse set of inputs, which counts as corruption $\delta_\times$ rather than as a drift ϵ. The last row also aborts on about half of its runs, because it compresses the output to a single bit.

duction rather than from one. Slightly more than s runs typically reach 99% success [74]. Shor's original algorithm, with no tradeoff, behaves differently. In Ekerå's analysis of it, a single run succeeds only with probability between about 60% and 82%, and this is a feature of the exact algorithm rather than of any approximation. Pushing this past 99% costs either a few extra point additions plus one bounded classical search, or two bounded classical searches in post-processing with no extra point additions [70].[12]

Remark A.12 (Recycling and compression are exclusive). The free input register and the compressed oracle do not combine for free. Recycling the control qubit (above) requires measuring each control as the

[12]The heuristic 60–82% range and the two routes past 99% are from Ekerå's own analysis [70, Abstract]. The range reflects the group order r and search-space size, and the extra operations sharpen the QFT peak in the same way the register size does.

oracle runs. The residue-arithmetic construction of Section A.7, by contrast, compresses the *work* register but needs its control registers intact, so it leaves them unrecycled [11, Sect. 1]. The tension is concrete in the compressed `ECDLP` construction [14]: it does not recycle its index registers, and spends its mid-circuit measurements instead on a *pebbling* schedule. A pebble game is a bookkeeping scheme. For each intermediate result of a reversible circuit, it records whether that result is kept in memory or freed and recomputed later. This trades space against gate count. *Spooky* pebbling [75, 76] is the variant that frees a result's qubits early by *measuring* it, and repairs the phase error this leaves behind at a later point. Here the results being pebbled are the nodes of a binary tree. Its leaves are the windowed point additions and its root is the accumulated projective point. Each node stores a residue triple in the RNS of Section A.7. The tree shape comes from how the point accumulation is structured. The affine constructions [6, 12, 13] make the opposite choice: they recycle the control qubit and use measurement only inside the arithmetic. A construction can thus collapse the input register or compress the output register. Obtaining both at once is open. This is the fault line along which the frontier of Table 7 splits.

ii From scalar multiplication to point additions. The oracle stage decomposes exactly as the modular exponentiation of Section A.3 did, one level at a time. Writing both exponents in windows of w bits, $a = \sum_k 2^{kw} a_k$ and $b = \sum_k 2^{kw} b_k$ with $0 \le k < \lceil n/w \rceil$, the point multiplication factors as

$$[a]P + [b](-Q) = \sum_k \big([a_k][2^{kw}]P + [b_k][2^{kw}](-Q)\big)$$

Collecting the $K = 2\lceil n/w \rceil$ window values in a single sequence $(i_0, \ldots, i_{K-1}) = (a_0, \ldots, a_{K/2-1}, b_0, \ldots, b_{K/2-1})$, the oracle reduces to $K \approx 2n/w$ *windowed point additions*

$$|i_k\rangle\,|R\rangle \;\longmapsto\; |i_k\rangle\,\Big|R + P_{i_k}^{(k)}\Big\rangle, \tag{61}$$

where the table of window k, $(P_0^{(k)}, \ldots, P_{2^w-1}^{(k)})$, holds the classically precomputed multiples $P_j^{(k)} = [j]\big([2^{kw}]P\big)$ for $k < K/2$ and $P_j^{(k)} = [j]\big([2^{(k-K/2)w}](-Q)\big)$ otherwise. The addend is loaded by a table lookup at a cost of 2^w Toffoli gates and unloaded almost for free by measurement-based uncomputation [77]. This is the same design philosophy as before: the multiples $[2^{kw}]P$ and $[2^{kw}](-Q)$ play the role of the constants $g^{2^i} \bmod N$, they are computed classically, and every window costs the same regardless of k.

Figure 21 shows the resulting accumulation stage. Block k reads its addend from table k, indexed by the window register $|i_k\rangle$, and adds it to the shared accumulator R with the point-addition circuit of Figure 35. The figure is drawn in the coherent form, with all K window registers present, that is $2n$ exponent qubits. In practice, the inverse quantum Fourier transform on the exponents is performed semi-classically [53]. The windows of each exponent are processed from the most significant one down, since each measured window sets the phase corrections of the less significant ones. Once a block is done, its w qubits receive classically controlled phase rotations, are measured, and are re-prepared for the next window. A single w-qubit register is therefore reused throughout, and the logical qubit counts of this section include only w exponent qubits.

The count K is itself only the naive baseline. Following Litinski [73], the very first addition, into the starting point $\mathcal{O}$, is just a lookup of the table point itself. This also avoids feeding the point at infinity to the affine addition circuit, which does not accept it as input, except when $i_0 = 0$, where the looked-up point is itself $\mathcal{O}$. The last three additions can be folded into extra classical post-processing, using the single-run, bounded-search mode of Ekerå's general-logarithm algorithm [70], instead of being run quantumly. For `secp256k1` with $w = 16$, this brings $K = 32$ down to the $N_A = K - 4 = 28$ windowed additions actually needed [6, 12, 13]. The addend is no longer a classical constant but a point read from the table, so the number of lookups per window depends on how much free space the surrounding circuit has left. The gate-oriented constructions can afford three such lookups [12]. The space-optimal one, with only a single free n-qubit register to work with, reports five [13], at $5 \cdot 2^w$ rather than $3 \cdot 2^w$ Toffoli gates per window.

iii From point additions to field arithmetic. A single controlled multiplication in Section A.3 still had to be built out of modular arithmetic. A single windowed point addition $R \mapsto R + P_i$ is no different. It still has to be built out of the affine addition law. Write the running accumulator as $R = (x_R, y_R)$ and the freshly loaded table point as $P_i = (x_i, y_i)$. In the generic case, the law reads

$$\begin{aligned} \lambda &= \frac{y_i - y_R}{x_i - x_R}, \\ x_{R'} &= \lambda^2 - x_R - x_i, \\ y_{R'} &= \lambda(x_R - x_{R'}) - y_R, \end{aligned} \tag{62}$$

where $R' = (x_{R'}, y_{R'})$ is the value written back into the accumulator. "Generic" means $R \neq \pm P_i$ and neither is $\mathcal{O}$. Doubling and the point at infinity use different formulas, but they occur with negligible probability once the accumulator is seeded at a nonzero point rather than at $\mathcal{O}$ itself. Both are therefore neglected here, as elsewhere in this literature. The first-addition lookup described above seeds the accumulator at $P_{i_0}^{(0)}$, which is itself $\mathcal{O}$ whenever $i_0 = 0$. That

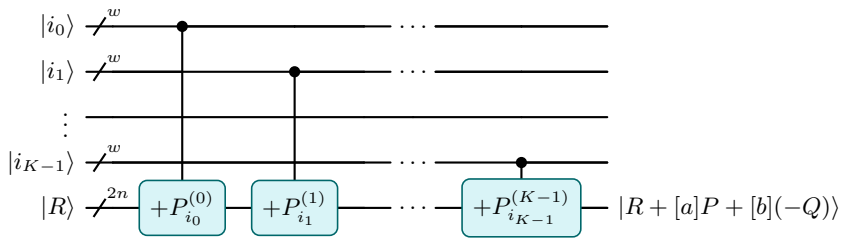


Figure 21: Windowed point-accumulation stage, in coherent form: block k adds to the accumulator R the table point $P_{i_k}^{(k)}$ selected by the window register $|i_k\rangle$, one instance of Equation (61) per window. In the implementation, a single w-qubit window register is recycled through the semi-classical Fourier transform.

branch has weight 2^{-w} per run, so it must be handled explicitly or counted as corruption. The case $i = 0$ is handled as the only special case [4, 12, 62]. This last case is not a matter of negligible probability like the other two. The window $(P_0, \ldots, P_{2^w-1})$ always sets $P_0 = \mathcal{O}$ by construction, so whenever a window's value happens to be 0 the addend is certainly $\mathcal{O}$. The circuit must therefore explicitly recognize this and leave R unchanged, rather than run the generic formulas above. Equation (62) fixes the registers a reversible circuit must hold. Besides the $2n$-qubit accumulator (x_R, y_R), which already holds the point accumulated so far, the circuit needs the table point (x_i, y_i) just loaded by Equation (61) and a workspace register for λ. Three more n-qubit registers are needed, which together with the accumulator and the w-qubit window index bring the qubit count in play to $2n + 3n + w = 5n + w$. Realizing the map $R \mapsto R'$ reversibly then means executing, in order,

subtract:	$\Delta x \leftarrow x_i - x_R$,
	$\Delta y \leftarrow y_i - y_R$
invert:	$\Delta x \leftarrow \Delta x^{-1}$
multiply:	$\lambda \leftarrow \Delta y \cdot \Delta x$
square, combine:	$x_{R'} \leftarrow \lambda^2 - x_R - x_i$
combine, multiply, combine:	$y_{R'} \leftarrow \lambda(x_R - x_{R'}) - y_R$
uncompute:	release Δx, Δy, λ, (x_i, y_i)

and Figure 22 lays the same sequence out as a dataflow diagram. (In practice **invert** and **multiply** are usually fused into a single divide primitive [4, 62]. They are kept apart here to show exactly where the inversion enters.) The uncompute step is not free, but it is cheap to route. The line of slope λ through R and P_i also passes through $(x_{R'}, -y_{R'})$, the unreflected third intersection point that R' is the negative of. Since R is overwritten in place by R', λ is recovered from R' and the table point P_i, which can be loaded again by lookup, as $\lambda = (y_{R'} + y_i)/(x_i - x_{R'})$. It is uncomputed by recomputing it this way rather than by keeping a separate garbage copy [4]. Of the five arithmetic steps, one is qualitatively different from the rest. Subtraction costs $O(n)$ gates, while multiplication, squaring and inversion all cost $O(n^2)$, as in the caption of Figure 22. What sets **invert** apart is its constant. It is the one place this sequence asks for a modular inversion rather than a multiplication, and as the next paragraph makes precise, that constant makes it the dominant cost.

Gouzien et al. report a total of $9n + w + 4$ logical qubits [4] for the unoptimized construction. The total counts this layer's registers together with the workspace of the extended Euclidean algorithm, described in the next paragraph. It is of the same order as the $9n + 2\lceil \log_2 n \rceil + 10$ already quoted for Roetteler et al. in Table 7. This is the baseline every direction below cuts down from, rather than a new number in its own right.

iv The inversion bottleneck. Of the operations in Figure 22, both the gate count and the qubit count of the whole construction are dominated by **invert**. It is realized through a modular inversion computed by a reversible extended Euclidean algorithm (EEA). Although inversion and the multiplications elsewhere in the same figure are both $O(n^2)$, the constant is significantly larger for inversion [62, 72, 73]. This is the structural difference with factoring. There, the innermost primitive was a modular *addition*. The direct, large-register pipeline made it cheap with the coset representation of Section A.3. The modern construction does so by simply working modulo small primes in the residue system of Section A.7. Here it is a modular *inversion*, and no comparable technique makes it free.

v How the inversion is actually computed: Kaliski's algorithm. The reversible EEA invoked above is, concretely, a *binary* extended Euclidean algorithm due to Kaliski [78], chosen because the ordinary Euclidean algorithm is awkward to make reversible. Reversible dividers exist, but inside the Euclidean loop each quotient has a data-dependent length, so a fixed circuit would have to provision every step for the longest possible quotient. Kaliski's algorithm replaces division by repeated halving, so every branch is selected by parity tests and one com-

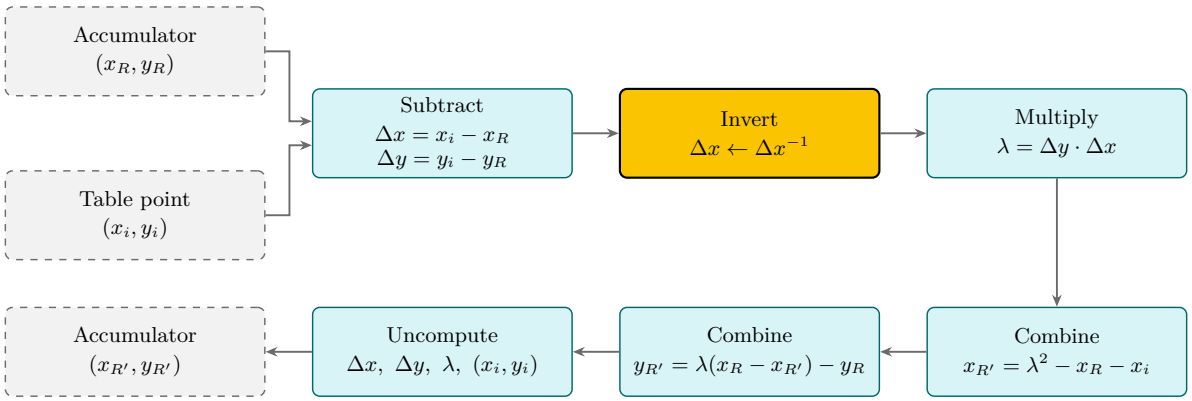


Figure 22: Textbook view of one windowed point addition $R \mapsto R + P_i$, following the affine addition law of Equation (62). The table point is loaded by a lookup (2^w Toffoli gates). The subtractions cost $O(n)$, while the inversion, the multiplications and the squaring all cost $O(n^2)$. In the circuits considered here, *invert* and *multiply* are fused into a single in-place division $\Delta y \leftarrow \Delta y/\Delta x$ that never materializes Δx^{-1}. The slope λ is not uncomputed separately. It is overwritten by a second in-place multiplication, $\lambda(x_i - x_{R'}) = y_{R'} + y_i$ (Figure 35). These two in-place multiplications account for most of the Toffoli count of the addition.

parison, and every step does the same fixed amount of work regardless of the input. We denote its four registers, (u, v, w, s), throughout.

The four registers are initialized to $(q, x, 0, 1)$ for computing $x^{-1} \bmod q$. The algorithm then applies the following *round* operation:

if u even: $u \leftarrow u/2, \quad s \leftarrow 2s$
elif v even: $v \leftarrow v/2, \quad w \leftarrow 2w$
elif $u > v$: $u \leftarrow (u - v)/2, \quad w \leftarrow w + s, \quad s \leftarrow 2s$
else: $v \leftarrow (v - u)/2, \quad s \leftarrow w + s, \quad w \leftarrow 2w$

Each round is one of four mutually exclusive branches, selected by the parities of u, v and a comparison. Each branch is a shift, an addition, or a conditional subtraction, so no division ever appears. Run classically, the loop stops as soon as $v = 0$. A quantum circuit cannot stop early on a data-dependent condition, so instead it runs a fixed $2n$ rounds for an n-bit modulus, an upper bound on how many rounds any input can need. The rounds after $v = 0$ are not automatically no-ops. With $v = 0$ the branch "v even" still fires and doubles w. Reversible implementations therefore add a termination flag, set once $v = 0$, that disables every later round, and a counter that records the number k of active rounds [72]. At termination $w \equiv -x^{-1}2^k \pmod q$, with $n \le k \le 2n$ depending on the input [78]. At this point $0 < w < 2q$ [4], so w is first reduced, $w \leftarrow w - q$ if $w \ge q$, and then negated, $w \leftarrow q - w$, giving $w \equiv x^{-1}2^k \pmod q$ with $0 < w < q$. Only then does a second pass strip the extra factors of two. It runs $2n$ further rounds of "if w even, $w \leftarrow w/2$, else $w \leftarrow (w + q)/2$", each controlled on the counter, so that exactly k of them act and the rest are disabled. This leaves $w = x^{-1} \bmod q$. Both passes are therefore circuits of fixed, input-independent length, built entirely from halvers, comparators, one conditional subtraction, a single negation, controlled adders, and the flag and counter logic, which is what made Kaliski's algorithm the basis of the earlier reversible constructions [4, 62, 72]. Gouzien et al. need neither the flag nor the second pass. They let the idle rounds keep doubling w, so that phase 1 always ends with $w \equiv -x^{-1}2^{2n} \pmod q$, and absorb the fixed factor 2^{2n} into the Montgomery representation [4]. Each of the three directions below pays for, avoids, or reorganizes exactly this inversion, and the two affine ones replace Kaliski's algorithm, by a different binary algorithm and by the quotient-based Euclidean algorithm respectively.

As an example, computing $3^{-1} \bmod 11$ starts at $(u, v, w, s) = (11, 3, 0, 1)$ and runs:

round	branch taken	u	v	(w, s)
1	$u > v$	4	3	$(1, 2)$
2	u even	2	3	$(1, 4)$
3	u even	1	3	$(1, 8)$
4	else	1	1	$(2, 9)$
5	else	1	0	$(4, 11)$

Round 5 reaches $v = 0$, so phase 1 halts after $k = 5$ rounds with $w = 4$. Since $3^{-1} \bmod 11 = 4$, the natural guess is $w \equiv x^{-1}2^k = 4 \cdot 2^5 \equiv 7 \pmod{11}$. Instead $w = 4$, and $4 \equiv -7 \pmod{11}$. This sign flip is not specific to this example: the recurrence as stated always computes $w \equiv -x^{-1}2^k \pmod q$, not $w \equiv x^{-1}2^k \pmod q$, and running the halving pass below on the unfixed w only carries the wrong sign through to the end rather than correcting it. The fix is a conditional subtraction of q followed by a single negation, $w \leftarrow q - w$, applied once phase 1 halts and before phase 2 begins. Here $w = 4 < 11$, so the subtraction does nothing, and $w \leftarrow 11 - 4 = 7$, which is now $\equiv x^{-1}2^k \pmod{11}$ as required. The subtraction is not idle in general. For $x = 5$ and $q = 7$, phase 1 ends with $w = 8 \ge q$. Phase 2 then strips the $k = 5$ powers of two by halving modulo q:

round	branch taken	w
1	else	9
2	else	10
3	w even	5
4	else	8
5	w even	4

After exactly k rounds, $w = 4 = 3^{-1} \bmod 11$, as required. A real circuit cannot know k in advance. It runs both passes for their full length of $2n$ rounds, and the flag and the counter disable the rounds past the true stopping point, here $2n - k = 3$ rounds in each pass. Without the flag, round 6 would take the branch "v even" and double w from $(4, 11)$ to $(8, 11)$.

A.13 Optimize the Gates: Splitting the EEA

The first affine direction reorganizes the inversion itself, borrowing an idea from the reversible Reed–Solomon decoders of [79]. Decoding a Reed–Solomon code also reduces to an EEA computation (finding an error-locator polynomial via a polynomial Bézout identity), and [79] needed a fully reversible, low-qubit version of it for an unrelated quantum algorithm. The ordinary extended Euclidean algorithm carries the Bézout coefficients alongside the running remainders, and it is keeping those coefficients coherent that makes it expensive to run reversibly. The trick is to split it into two cheaper passes that never store the coefficients at all. The *Euclidean pass* runs the plain, non-extended algorithm on $(u, v) = (q, x)$ and records only which branch it takes at each iteration, as a string of *garbage* bits g. In the variant of [12], each iteration records two bits, $b_0 = v \bmod 2$ and $b_0 \wedge b_1$ with $b_1 = [u > v]$. This is a different binary Euclidean algorithm from Kaliski's, with its own Bézout coefficients, which we denote (r, s). The *reconstruction pass* then replays those recorded choices to rebuild the inverse. (The record g is what is called a *dialog*). Three properties make the split profitable:

(i) *Space.* The Euclidean pass runs $\approx 1.413n$ iterations and emits on average 1.5 garbage bits per iteration, so g is about $1.413n \times 1.5 = 2.12n$ bits long after the pass. The exact length of the garbage depends on two factors: the exact number of iterations, and the exact number of garbage bits emitted per iteration. The standard deviation of the iteration count around the average is of order $O(\sqrt{n})$, and one must actually run $1.413n + O(\sqrt{n})$ iterations to succeed with constant probability, where the constant in the O determines the chance of success. The second factor is removed in [12] by an *encoding-decoding* method which packs the bits created by three iterations into 5 bits deterministically. The register thus has $2.355n + O(\sqrt{n})$ qubits.

(ii) *In-place multiplication for free.* The Bézout updates are linear operations controlled by the recorded bits, so starting the reconstruction pass from $(r, s) = (y, 0)$ ends it at $(0, yx \bmod q)$, which is how the in-place multiplications of the point addition are implemented [12, Algs. 3–4]. Running the reconstruction pass backwards from $(r, s) = (0, y)$ ends it at $(yx^{-1} \bmod q, 0)$. The inversion and one modular multiplication thus come out of a single circuit [12, 79].

(iii) *No expensive reverse pass.* When a reversible binary EEA is run, the entire algorithm (including updates on (u, v) and (r, s)) runs once forwards and once backwards. In the new variant, the updates on (r, s) only need to be computed once, in the forward pass. Only the updates on (u, v) (conversion into dialog, and then back to the initial value) are run twice, removing the cost of the extensive reverse pass on (r, s).

The qubit count is maximal during the reconstruction step, since it must hold the dialog record g together with the two n-bit modular integers it combines. With the ideal g this is $2.12n + 2n = 4.12n$ qubits. With the fixed packed register actually implemented, it is $2.355n + 2n \approx 4.36n$ [12]. Figure 23 illustrates the same split on Kaliski's algorithm, whose four-branch rounds can be recorded and replayed in the same way. Its invariant differs from that of [12]. Replaying from Kaliski's initial $(w, s) = (0, 1)$ reproduces his phase-1 output, not the in-place multiplication of item (ii), so the figure shows the split itself.

A last factor of improvement on this direction comes from allowing the point addition to be inexact. All comparisons inside the Euclidean pass and inside the modular adders are performed on the top 40 to 50 bits only. The number of EEA iterations is fixed at $1.413n + c_{\text{iter}}\sqrt{n}$ with $c_{\text{iter}} \approx 2.4$ rather than being data-dependent. The margin $c_{\text{iter}}\sqrt{n}$ is a tail bound, not a tuning parameter. The number of iterations this algorithm needs on a random input is empirically normal, with mean $1.413n$ and standard deviation $\approx 0.6\sqrt{n}$. The constant $c_{\text{iter}} \approx 2.4$ is four such standard deviations, chosen so that an input needing more than the fixed length is a four-sigma event, about 3×10^{-5} per addition under the normal approximation. That is small, but of the same order as the measured failure bound below. A fixed length is needed for the reason met with Kaliski's algorithm above, since a reversible circuit cannot stop on a data-dependent condition, but Kaliski's $2n$ rounds are a worst-case guarantee whereas this length is a tail bound. When q is a pseudo-Mersenne prime $2^k - f$ with $f \ll 2^k$, as for `secp256k1`, a controlled subtraction of q further degenerates into erasing the most significant bit and adding the small constant f into the low bits. The composite circuit succeeded on all 10 000 random inputs tested in [12], which reports

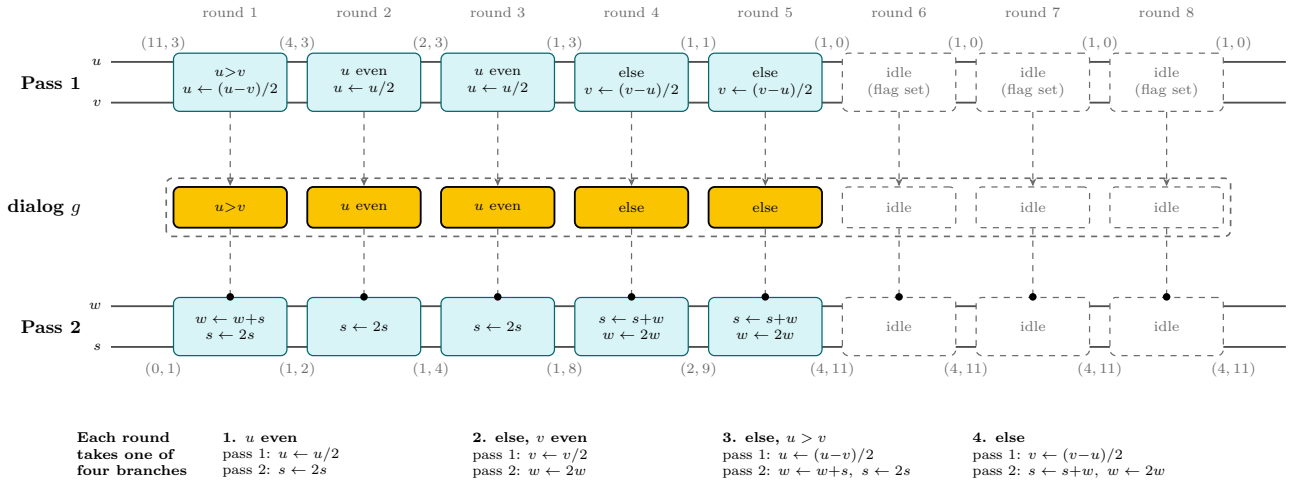


Figure 23: Unrolled trace of the dialog-split extended Euclidean algorithm (EEA), applied to Kaliski's algorithm on the worked example of Section A.12.0.v ($x = 3$, $q = 11$). Each round takes one of the four branches listed below the trace. Pass 1 updates (u, v) and records only the branch taken, in the dialog g (dashed frame). Pass 2 replays g on (w, s) and never revisits (u, v). Both passes run the fixed $2n = 8$ rounds of a reversible circuit. Once $v = 0$, after round 5, the termination flag makes the remaining rounds idle. Pass 2 ends at $(4, 11)$, Kaliski's phase-1 output, where the direct computation applies the negation $w \leftarrow q - w$. That negation and the halving phase are unchanged by the split and are not shown. The variant of [12] records different bits, and running its replay backwards from $(r, s) = (0, 1)$ ends at $(x^{-1} \bmod q,\ 0)$ directly.

this as a failure probability of at most $2^{-13.3}$, the reciprocal of the number of trials. This is an empirical observation, not a bound. Zero failures in 10^4 trials give a 95% upper confidence bound of about $3 \times 10^{-4} \approx 2^{-11.7}$ per addition (the rule of three). Comparable rates are reported in [6].

It is tempting to record this failure rate as an ϵ, but it is not one. Recall that Definition A.6 splits imperfection along two axes: ϵ bounds how far the oracle's values may drift from an exactly periodic function, while δ bounds the probability that the read-out R fails, split into an abort rate $\delta_\perp$ and a corruption rate $\delta_\times$. Condition (i), the one that fixes ϵ, is a deterministic, worst-case statement: $d_S(f(g), f_0(g)) \le \epsilon/2$ for *every* $g \in G$. An approximate point addition does not drift: on almost every input it returns the correct point, and on a sparse subset it returns an unrelated one. Curve points carry no modular distance, and with the discrete metric condition (i) fails for every $\epsilon < 2$, that is, for every non-trivial ϵ. The failure must therefore be accounted on the other axis, as a corruption $\delta_\times$ of the read-out. The distinction is not cosmetic: an ϵ-drift is a property of the oracle that no read-out can remove, and masking converts it into aborts and corruptions at rates about S and ϵ/S (Section A.8), whereas a sparse circuit failure is suppressed by making the circuit better and admits no read-out remedy at all. With 28 windowed additions per run [6, 12], a union bound on the 95% confidence bound puts the failing inputs at a fraction $p \lesssim 28 \cdot 3 \times 10^{-4} \approx 2^{-6.9}$ of the superposition, per run. Charging this weight as a corruption rate, $\delta_\times \lesssim p$, is the heuristic of Section A.8. The failing branches lower the overlap with the ideal state to no less than $1 - p$, so the infidelity is at most $2p \approx 2^{-5.9}$, and the rigorous trace-distance bound is only $\delta_\times \le \sqrt{2p} \approx 0.13$. The point estimate of p is lower, but it rests on zero observed failures. Please note that register sharing is also used in this EEA variant: the bits of g share the same register as the bits of (u, v) during dialog computation and uncomputation.

A.14 Shrink the Register: Sharing the EEA Exactly

The second affine direction returns to an original observation of Proos and Zalka [71]. Along the EEA, the sequence $\{r_i\}$ decreases while the paired Bézout coefficient $\{t_i\}$ increases, and they satisfy

$$r_{i-1}t_i + r_i t_{i-1} = q. \tag{63}$$

The two may therefore share a register.[13] Luo et al. [13] push this to a single $(n + 2)$-qubit register holding the *triple* (r_{i-1}, t_i, q_i). The register also holds the intermediate states that arise while computing q_i and updating r_{i-1}, t_{i-1}. A second $(n + 2)$-qubit register holds the complementary pair (r_i, t_{i-1}), written (r', t') below to distinguish it notationally from the first register. Three ingredients make it explicit and exact:

[13] Here r_i, t_i, q_i follow Luo et al.'s own notation and are unrelated to the curve order r used throughout Section A.12 or to the modulus q of Kaliski's algorithm above. We have $r_0 := q$, $r_1 := x$, and each step divides r_{i-1} by r_i to get the quotient $q_i = \lfloor r_{i-1}/r_i \rfloor$ and the next remainder $r_{i+1} = r_{i-1} - q_i r_i$. In parallel, $t_0 := 0$, $t_1 := 1$ and $t_{i+1} = t_{i-1} + q_i t_i$. When $r_k = 1$, the inverse is then $(-1)^{k-1} t_k \bmod q$ (a sign correction analogous to Kaliski's final negation above).

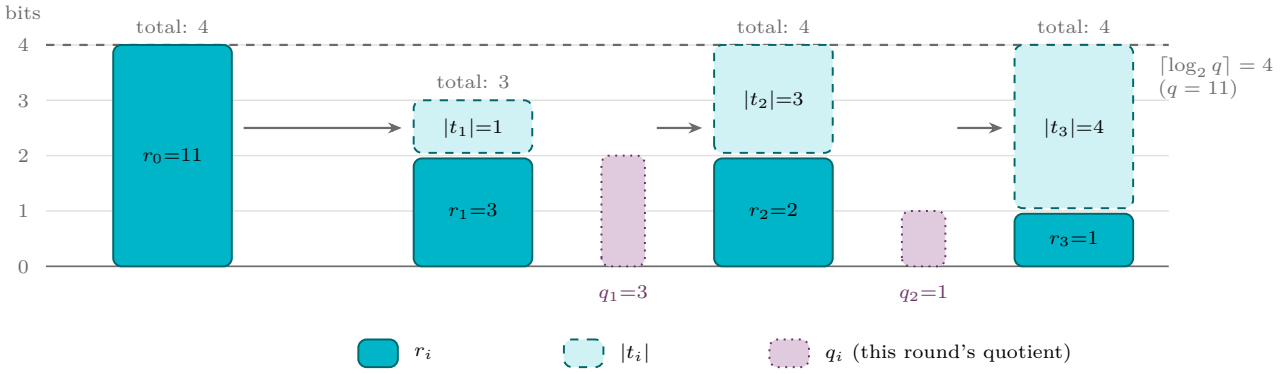


Figure 24: Register sharing in the quotient-based EEA of Section A.14, on $q = 11$ and $x = 3$, with r_i, t_i and q_i as defined there. Bar heights are bit-lengths. As r_i shrinks, $|t_i|$ grows into the qubits it frees. By invariant (63), $r_i|t_i| < q$, so the two bit-lengths sum to at most $\lceil \log_2 q \rceil + 1$. The tag above each column shows that here they never exceed 4. Each quotient $q_i = \lfloor r_{i-1}/r_i \rfloor$ is drawn just before the column it produces. The last one, $q_3 = 2$, gives $r_4 = 0$ and ends the algorithm. The figure shows bit-lengths only, not the layout of the triple (r_{i-1}, t_i, q_i) in the register of [13].

(i) *Length registers.* Four auxiliary registers of $\lfloor \log_2 n \rfloor + 1$ qubits each hold the field boundaries $\ell_t, \ell_q, \ell_{r'}$ and the shift counter ℓ_s. The shift counter is not a field length. It is the number of positions by which the second register is currently circularly shifted relative to the first, which keeps the two registers' fields aligned during arithmetic between them. Together they let one circuit serve inputs whose field lengths differ in superposition.

(ii) *Location-controlled arithmetic.* Additions, subtractions, swaps and length updates act only on the positions selected by those registers. This is realized by unary iteration [77] over a *step-dependent active window* $[k(T), K(T)]$. Here T denotes the step index, a global counter over all steps of the algorithm (different inputs x take different numbers of steps through the same fixed schedule). The bounds $k(T)$ and $K(T)$ are derived analytically over every possible input. They specify which register positions can possibly be touched at step T. Bounding the total number of steps means bounding how many iterations the classical Euclidean algorithm itself can take on an n-bit input, a question solved by Lamé's theorem. The slowest case occurs when every quotient along the way equals 1, forcing the remainders to shrink as slowly as possible. This happens exactly when they form a run of consecutive Fibonacci numbers. Consecutive Fibonacci numbers grow by the golden ratio $\varphi = \frac{1+\sqrt{5}}{2}$ at each step, so the worst case takes $c = 1/\log_2 \varphi \approx 1.44$ quotient steps per bit. Each iteration i of the extended Euclidean algorithm runs through four phases, and each phase takes one circuit step per bit of the quotient q_i. The iteration therefore costs $4(\lfloor \log_2 q_i \rfloor + 1)$ steps, and the total is $N_{\text{step}} = 4\sum_i(\lfloor \log_2 q_i \rfloor + 1)$ [13, App. A.1]. Luo et al. prove $4n \le N_{\text{step}} \le 4\lceil cn \rceil$. The lower bound holds because the steps count quotient bits, not quotients. An input that finishes in few iterations has large quotients, and their bit lengths still add up to about the bit length n of the modulus. The upper bound is the Fibonacci worst case above, where every quotient equals 1. At most $4\lceil cn \rceil \approx 5.76n$ steps of $O(n)$ gates are therefore needed.

(iii) *Exact allocation.* Where Proos and Zalka used a uniform worst-case allocation, incurring $O(\sqrt{n})$ overhead and truncating rare outliers at a cost in fidelity, this scheme guarantees a deterministic register bound for every input, reducing the overhead to $O(\log n)$ while preserving exact correctness.

The inversion then fits in $2n + 6\lfloor \log_2 n \rfloor + 19$ qubits and $217n^2 + O(n \log_2 n)$ Toffoli gates.[14] We illustrate this technique in Figure 24.

This direction reaches $3n + O(\log n)$ for the whole point addition, rather than the $4.36n$ of the dialog split, by one further trick worth stating because it is a second and quite different use of mid-circuit measurement. Besides the two n-qubit registers R_x, R_y holding x, y, the in-place division $|x\rangle |y\rangle |0\rangle_A |0\rangle_S \mapsto |x\rangle |y/x\rangle |0\rangle_A |0\rangle_S$ uses an n-qubit workspace register A and a small $O(\log n)$-qubit register S. The forward EEA runs on R_x using A purely as scratch space, in the same sense as the (u, v) wires of Figure 23: overwritten and discarded, never the thing that has to be undone later. Concretely, A returns to $|0\rangle$ once x^{-1} has been written into R_x. The leftover state that cannot yet be uncomputed is parked in S, playing the role g played there. R_y is untouched throughout and still holds the original y. A separate modular multiplication then writes y/x into A, and only at this

[14] These figures are from version 2 of [13] (arXiv:2604.02311v2, 9 August 2026). Version 1 (2 April 2026) reported $3n + 4\lfloor \log_2 n \rfloor + O(1)$ qubits and $204n^2 \log_2 n + O(n^2)$ Toffoli gates for the same inversion.

point does A become dirty. The resolution is that the original y, still sitting in R_y, has become redundant. It is recoverable from x and y/x. One therefore measures R_y in the X basis and resets it. This yields a uniformly random bitstring $b \in \{0,1\}^n$ and leaves only the phase $(-1)^{b\cdot y}$. R_y is now clean and serves as workspace for the inverse EEA. The inverse EEA uses it to clear S and recover x in R_x. The phase is removed in three steps. First, y is recomputed into R_y from x and y/x. Second, the correction $Z^b = \bigotimes_k Z^{b_k}$ is applied, that is, a Pauli Z gate on each qubit k with $b_k = 1$. Third, the recomputation is uncomputed. Swapping R_y and A then completes the map [13]. It is exact, which is why it contributes no relaxation to the (ϵ, δ) accounting at all.

A.15 Avoid the Inversion: Projective Coordinates and a One-Bit Output

The third direction is the construction of Chevignard, Fouque and Schrottenloher [14]. It is the only one that leaves affine coordinates behind. In doing so, it is also the only one that reuses the factoring optimizations. It fuses the output-register compression of Section A.6 with the residue arithmetic of Section A.7. Two observations drive it. First, the inversion exists only because affine coordinates represent a point by $(x, y) = (\mathcal{X}/\mathcal{Z}, \mathcal{Y}/\mathcal{Z})$ for some $\mathcal{Z} \neq 0$. Reconstructing x, y from $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ is exactly where the division above came from. *Projective* coordinates instead keep the triple $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ itself, without ever dividing. The chord-and-tangent addition law then rewrites into polynomial formulas in $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ that use only multiplications and additions in $\mathbb{F}_q$ [80, 81]. The point can thus be advanced with modular multiplications alone. Those multiplications are then accumulated in a residue number system, exactly as the unreduced product $\prod_k M_k^{e_k}$ was in Section A.7, so every intermediate value occupies only $O(\log \log q)$ qubits. Second, reading the answer out would normally mean converting the projective point back to affine form, which reintroduces a division (recovering $x = \mathcal{X}/\mathcal{Z}$). This is sidestepped by never forming the point at all, and instead outputting a *single bit*, the Legendre symbol of the affine x-coordinate. It needs no inversion. With $x = \mathcal{X}/\mathcal{Z}$, multiplicativity gives $\left(\frac{\mathcal{X}\mathcal{Z}}{q}\right) = \left(\frac{x\mathcal{Z}^2}{q}\right) = \left(\frac{x}{q}\right)$, so the bit is computed from $\mathcal{X}\mathcal{Z} \bmod q$ and is the same for every projective representative of the point [14]. The difference between the affine and projective coordinate computations for point addition is depicted in Figure 25.

Compressing the output to one bit is exactly the May–Schlieper move of Section A.6, and it is justified here by three facts, which we take in turn. First, May–Schlieper [9] guarantees that period finding survives compression to t bits at the cost of a factor $1/(1-2^{-t})$ in the number of runs, so a single output bit ($t = 1$) merely doubles the run count. This doubling applies to the run count Ekerå's general-logarithm tradeoff [70] already needs for its own lattice-based post-processing, not to a single-shot baseline. For $n = 256$, that tradeoff already targets 11 measurements to reach 99% post-processing success, and doubling gives exactly the 22 runs quoted below, matching the pattern at every curve size [14]. Second, that guarantee was stated for a hash drawn at random from a universal family, which the Legendre symbol is not. Chevignard et al. randomize it by adding a random multiple uP to the point before taking the symbol, and treat the resulting family as close enough to universal under an explicit heuristic [14, Heuristic 1]. Results on the balance of character sums over multiples of a point support that heuristic [82, 83], but do not prove it. The hashing step is therefore heuristic, unlike the May–Schlieper theorem it borrows. Third, the Legendre symbol must itself be computed reversibly. The Legendre symbol is the special case of the *Jacobi* symbol where the lower argument is prime, so a binary Jacobi-symbol algorithm [84] computes it directly, with no adaptation needed, and its structure and garbage accounting mirror the Euclidean passes above. Exceptional cases in the projective addition law are handled with complete formulas [80, 81]. The residue reconstruction here is not the truncated one of Section A.7. A Legendre symbol needs the exact residue $\mathcal{X}\mathcal{Z} \bmod q$, not its top bits, and an arbitrarily small error can flip it. The reconstruction therefore uses explicit Chinese remaindering, with its precision set so that it is exact except with negligible probability over the input [14, Lemma 2]. The Jacobi-symbol circuit's iteration count is similarly fixed so that it succeeds with probability at least 0.99 [14, Heuristics 3–4, Lemma 4]. By the criterion applied to the approximate point addition above, a computation that is exact except on a sparse set of inputs has $\epsilon = 0$, and its failures count as corruption $\delta_\times$. Masking has no role here. The output is already a single bit, and there is no value drift for it to absorb.

The resulting algorithm uses $3.12n + o(n)$ qubits at the price of raising the gate count from $O(n^3)$ to $\widetilde{O}(n^4)$. For $n = 256$ the estimate is 1193 logical qubits[15] and 22 independent runs of $2^{38.98}$ Toffoli gates each [14].

A.16 Trade-offs for ECDLP

No single algorithm dominates for ECDLP. Space, gate count and depth are pushed by three different lines of work, the depth axis being held by the low-depth point-addition circuits of Kim et al. [85]. More importantly, the strategy that dominates for RSA does *not* survive here. The compressed con-

[15] We quote the full version of [14] (IACR ePrint 2026/280), which accompanies the EUROCRYPT 2026 paper. Its abstract and Table 8 agree on 1193 qubits, 22 independent runs, and $2^{38.98}$ Toffoli gates for $n = 256$.

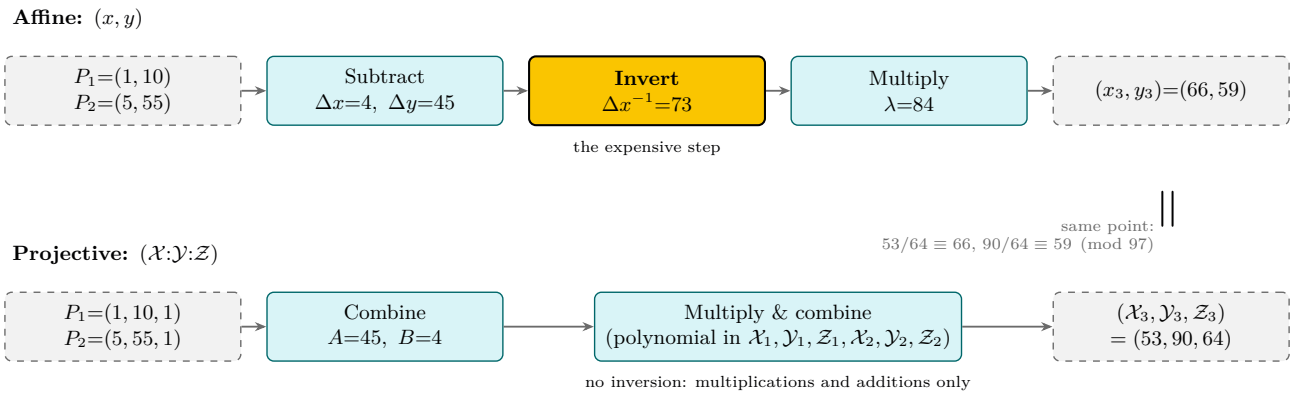


Figure 25: The same point addition in affine and in projective coordinates, on the curve $y^2 = x^3 - 3x + 5$ over $\mathbb{F}_{97}$, with $P_1 = (1, 10)$ and $P_2 = (5, 55)$. The affine law needs the inverse of Δx. The projective law needs only multiplications and additions, and its output $(\mathcal{X}_3 : \mathcal{Y}_3 : \mathcal{Z}_3)$ decodes to the same affine point $(66, 59)$. The projective row shows the structure of the computation. The construction of [14] uses the complete addition formulas of [80, 81].

Work	**Qubits** (n)	**Qubits** (256)	**Toffoli**
Proos–Zalka [71]	$5n + o(n)$	—	—
Roetteler et al. [72]	$9n + 2\lceil \log_2 n \rceil + 10$	2330	$\approx 2^{36.9}$
Häner et al. [62]	$8n + 10.2 \log_2 n + O(1)$	2124	$\approx 2^{30}$
Litinski [73]	—	≈ 2350 [a]	$2^{27.57}$
Babbush et al. [6]	—	1191 / 1441	$2^{26.27}$ / $2^{25.94}$
Schrottenloher [12]	$4.36n + O(\sqrt{n})$	1208 / 1462	$2^{26.11}$ / $2^{25.78}$
Chevignard et al. [14]	$3.12n + o(n)$	1193	$2^{38.98}$
Luo et al. [13]	$3n + 6\lfloor \log_2 n \rfloor + O(1)$	835	$2^{30.88}$

Table 7: Logical-level cost of solving ECDLP for $n = 256$ (curve secp256k1 for [6, 12, 13], P-256 otherwise). Qubit and Toffoli counts are for the whole algorithm, per run, including the windowed point additions. For [6, 12] they are taken from [12, Table 2]. The asymptotic formulas may count a single point addition only. Two-figure entries are the space- and gate-optimized variants of the same construction. The last two rows are the *avoid the inversion* and *shrink the register* directions of Section A.12. A point addition is called *exact* here if it uses no approximate comparisons and no truncated iteration count, so that it is correct on every input apart from the exceptional cases of the addition law, which the affine constructions neglect and the complete formulas of [14] handle. The point additions of [13, 62, 72] are exact. Those of [6, 12] truncate the Euclidean iteration count and fail on a small, empirically measured fraction of inputs. Run counts, circuit depth, and physical-level estimates are omitted (discussed above). [a] Litinski's paper does not report a comparable logical-qubit count, since it uses an active-volume cost model. The figure ~ 2350 is Schrottenloher's own comparison, "about twice as much qubits" as the space-optimized figures of Babbush et al. and of his own work [12], not a number from Litinski [73] itself.

struction of [14] was, on publication, the most space-efficient polynomial-time ECDLP algorithm known. It has since been dominated on *both* axes at once by the register-sharing construction, using 835 rather than 1193 logical qubits *and* $2^{30.88}$ rather than $2^{38.98}$ Toffoli gates [13] (Table 7). Asymptotically the same holds: $3n+O(\log n)$ qubits with $1008\, n^3/\log_2 n+O(n^2)$ gates against $3.12n + o(n)$ with $\widetilde{O}(n^4)$.

The reason is visible in the preceding paragraphs. In factoring, residue arithmetic eliminates the register that dominated the count, the n-bit exponentiation accumulator, and nothing cheaper was available. In ECDLP the dominant register is the EEA workspace of the inversion, and *two* reorganizations of the EEA remove it while staying inside $O(n^3)$. Compression, by contrast, buys the same space by a much more expensive route, and additionally forfeits the free input register of Remark A.12. The Pareto frontier now has five points, all on the affine branch. Ordered by qubit count, they are [13] (835 qubits, $2^{30.88}$ Toffoli gates), then the space- and gate-optimized variants of [6] and of [12], which trade qubits for gates down to 1462 qubits and $2^{25.78}$ Toffoli gates. This comparison mixes curves. The figures of [6, 12] are for secp256k1, whose special prime makes the arithmetic cheaper. For a generic prime, Schrottenloher's variants cost $2^{26.66}$ and $2^{26.33}$ Toffoli gates instead [12, Table 2]. For context, the earlier affine estimates that these improve upon are Proos–Zalka's $5n + o(n)$ qubits [71], and the point-addition figures of Roetteler

et al. [72] and Häner et al. [62] (see Table 7).

Two things are worth flagging about Table 7. Its concrete qubit counts are each taken from the source's own detailed per-curve estimate, never obtained by substituting $n = 256$ into the asymptotic formula quoted in the same row. For Chevignard et al., that substitution would give $3.12 \times 256 \approx 799$, well short of their reported 1193, because the suppressed $o(n)$ term is not yet small at this size. Litinski's own headline figure of 50 million Toffoli gates assumes an active-volume architecture with batching across many keys. The table instead reports $2^{27.57}$, the plain single-run Toffoli count of the same underlying circuit as given in [12], for comparability with the other rows. Circuit depth is a further axis the table omits. See Kim et al. [85] for the lowest-depth point-addition circuits, and [4, 6] for physical-level cat-qubit and surface-code estimates. These last are architecture-specific and not comparable across rows.

B Logical Implementation of Shor's Algorithm

This appendix turns the algorithms of Appendix A into explicit logical circuits. Each algorithm is first written as classical pseudo-code, then compiled into reversible arithmetic built from a few primitives, namely adders, comparators, table lookups and their measurement-based uncomputation. We focus on the part that dominates the cost of each attack. For factoring and FFDLP, this is the approximate modular exponentiation of Section A.9, detailed in Section B.1. For ECDLP, it is the elliptic curve point addition and, within it, the modular inversion, detailed in Section B.2. Every subroutine that produces intermediate values needs a matching uncomputation, so that its work registers return to $|0\rangle$ before the final measurement. We therefore describe each subroutine together with its adjoint, when they are not the direct inverse function. The semiclassical quantum Fourier transform is implemented as in [11, App. A.5 and Fig. 15]. We do not detail it further, since its cost is negligible next to that of the oracle.

i Software Implementations in Qualtran and Qarton. All exponentiation and compressed exponentiation circuits considered in this paper, whether in the RSA, FFDLP or ECDLP cases, are classical arithmetic circuits which, on a computational basis state, produce a computational basis state. Furthermore they can be entirely decomposed into purely classical operations (X, CX, CCX) as well as phase operations (Z, CZ, CCZ) and classically controlled operations following from measurement-based uncomputation. Additionally, the analysis can be reduced to sub-circuits with modest sizes, either a single residue computation for RSA and FFDLP, or a single elliptic curve point addition for ECDLP. It then becomes natural to entirely implement these circuits (i.e. construct them explicitly) and to assert their correctness by simulating them on classical inputs. The combination of classical operations and phase operations, with classical controls, produces the so-called *kickmix* circuit class [6], which can be efficiently simulated.

For RSA and FFDLP, Chevignard et al. [10] provided partial implementations in Qiskit [86]. For ECDLP, Schrottenloher [12] provided an implementation in Qarton. For their space-efficient version, Luo et al. [13] provided a Qiskit implementation. The new implementations we provide use Qualtran [41] and Qarton [40, 87]. Both libraries offer a high-level programming interface that abstracts the exact memory layout of a quantum circuit away from the programmer. In Qarton, the circuit is then constructed explicitly. Qualtran can likewise compile a program down to an explicit circuit, but its resource estimates are typically computed hierarchically from the program's call graph without doing so. The resulting qubit and gate count estimates are precise, but hold only in the logical circuit model, without taking into account routing, distance between logical qubits, etc. More precisely, we give:

- A Qarton implementation of the RSA and FFDLP compressed-exponentiation routine of [10], with the improvements by Gidney [11]. The exact gate and qubit counts obtained are similar, but differ slightly due to low-level optimization choices.
- A Qualtran implementation of the RSA and FFDLP of the modern algorithm as described in Section A.9.

and use existing implementation in Qarton for the ECDLP case.

All source code used can be found at [88] for Qarton and [89] for Qualtran.

B.1 Approximate Modular Exponentiation for Factoring

This section provides a detailed low-level description of the implementation of Shor's oracle function for the factoring problem. We follow the implementation presented in Section A.9 which is described here in detail for completeness. More precisely, this section focuses on the oracle that computes the approximate modular exponentiation whose high-level circuit can be seen in Figure 15. For the sake of clarity, Table 8 recalls all the parameters used in the modern Shor algorithm whose notation we rely on, throughout this appendix. In order to properly implement this algorithm at the logical level, we decompose the oracle function, namely the computation of the approximate modular exponentiation, into four subroutines. Each of these subroutines is then decomposed into arithmetic blocks (adders, lookup tables, etc.), from which we can compute the cost of the overall computation. An overview of the circuit, decomposed into 4 main subroutines, is presented in Figure 26. In the next paragraphs, we detail the subroutines:

- **ComputeExponent**: computes S_i, the logarithm of the residue r_i. The subroutine combines the uncomputation of S_{i-1} with the computation of S_i. The last call to the subroutine (outside of the loop) is used to uncompute the last exponent, returning the register to $|0\rangle$.
- **ReduceExponent**: computes the modular reduction for the exponent output of ComputeExponent. It also produces the quotient register ($\log(n_e)$ qubits, Section B.1.2.iii), which ReduceExponent† needs to undo the reduction.
- **ComputeResidue**: computes the residue with a small modular exponentiation.

Symbol	Definition	Role	Symbol	Definition	Role
N	integer to factor	classical	g_{p_i}	generator of $\mathbb{Z}^*_{p_i}$	classical
g	base of the exponentiation	classical	M_k	$g^{2^k} \bmod N$	classical
e	exponent, in uniform superposition	quantum	$D_{p_i,k}$	discrete log of $M_k \bmod p_i$	classical
V	exponentiation modulo N	quantum	r_i	residue for the prime p_i	quantum
n_e	size of the exponent register	classical	S_i	logarithm of r_i	quantum
p_i	i-th prime of the RNS	classical	$C_{j,k}$	truncated CRT factors	classical
$\lvert P\rvert$	number of primes in the RNS	classical	Acc_i	partial sum over the first i primes	quantum
ℓ	bit length of an RNS prime	parameter	w_1	window, discrete-log computation	parameter
κ	size of the accumulation register	parameter	w_{3a}	window, exponentiation outer loop	parameter
$\bar{\kappa}$	bits removed by the truncation	parameter	w_{3b}	window, exponentiation inner loop	parameter
			w_4	window, accumulation	parameter

Table 8: All parameters used in the modern Shor algorithm presented in Section A.9.

- **AccumulateResidue**: adds the residue to the accumulation register to build the approximate modular exponentiation $\tilde{V}$. All necessary phase corrections are included in the subroutine.

Algorithm 1 lists them in the order in which the oracle calls them, with $D_{p_{-1},k} = 0$ for the first prime. Two subroutines are not unitary on their own, so their adjoints need their own definition. ComputeResidue measures some of its work qubits in the X basis and defers the resulting phase corrections. Its adjoint ComputeResidue† is therefore a separate circuit, a left inverse that satisfies ComputeResidue$^\dagger \circ$ ComputeResidue = id. ReduceExponent maps S_i to $S_i \bmod (p_i - 1)$, which on its own loses information, since many values of S_i share the same remainder. The circuit makes the reduction reversible by also producing the quotient register (Section B.1.2.iii). ReduceExponent† is therefore not the adjoint of the reduction alone. It inverts the whole circuit, and it needs the quotient register, kept until then, to restore S_i.

Each subroutine is decomposed into arithmetic operations, which serve as the main building blocks for the entire circuit. There are three main operations:

- **Adder**: addition, subtraction, and comparison operations rely on circuits that are very similar in structure and cost. When two quantum registers are added together, we denote it a Q-Q Adder (quantum-quantum adder), and when one of the two operands is classical, a Q-C Adder (quantum-classical adder). A similar notation is used for comparison operations, namely Q-Q and Q-C Comparison. A subtraction is the adjoint of an addition and has the same cost. Every addition is performed modulo 2^n, where n is the size of the input number.

- **Lookup**: given an *address register*, loads into a *target register* a value precomputed classically and associated to each address value. Its cost is directly correlated to the two input parameters. The address size determines the Toffoli cost and impacts the qubit cost. The target register size only impacts the qubit cost.

- **Phaseup**: applies to an address register a phase given by a classical table. Combined with measurement-based uncomputation [90], it uncomputes Lookup operations efficiently. [16]

Algorithm 1 Approximate modular exponentiation oracle.

Input: exponent register $|e\rangle$, accumulator A on $\kappa+1$ qubits, prepared in the mask state $|\sigma\rangle$
Output: $|e\rangle$, and A shifted by $\tilde{V}(e) \approx (g^e \bmod N) \gg \bar{\kappa}$

1: $S \leftarrow 0$ ▷ $(\ell + \log n_e)$-qubit register
2: **for** $j = 0$ to $|P| - 1$ **do**
3: $S \leftarrow S + \sum_k e_k \left(D_{p_j,k} - D_{p_{j-1},k}\right)$ ▷ ComputeExponent, $S_{j-1} \mapsto S_j$
4: $(S, a) \leftarrow \left(S \bmod (p_j - 1),\ \lfloor S/(p_j - 1)\rfloor\right)$ ▷ ReduceExponent
5: $r \leftarrow g_{p_j}^S \bmod p_j$ ▷ ComputeResidue, ℓ-qubit register
6: $A \leftarrow A + \sum_k r_k C_{j,k} \bmod (N \gg \bar{\kappa})$ ▷ AccumulateResidue
7: uncompute r ▷ ComputeResidue†
8: $S \leftarrow S + a\,(p_j - 1)$, uncompute a ▷ ReduceExponent†
9: **end for**
10: $S \leftarrow S - \sum_k e_k D_{p_{|P|-1},k}$ ▷ ComputeExponent, $S \leftarrow 0$
11: apply the deferred phase corrections on $|e\rangle$ ▷ Phaseup

Table 14 gives the operation counts of every subroutine, and Table 15 its qubit counts.

[16] Uncomputation of multiple Lookups that share the same address register can be merged into a single Phaseup as long as the address register is only used as control qubits before the Phaseup.

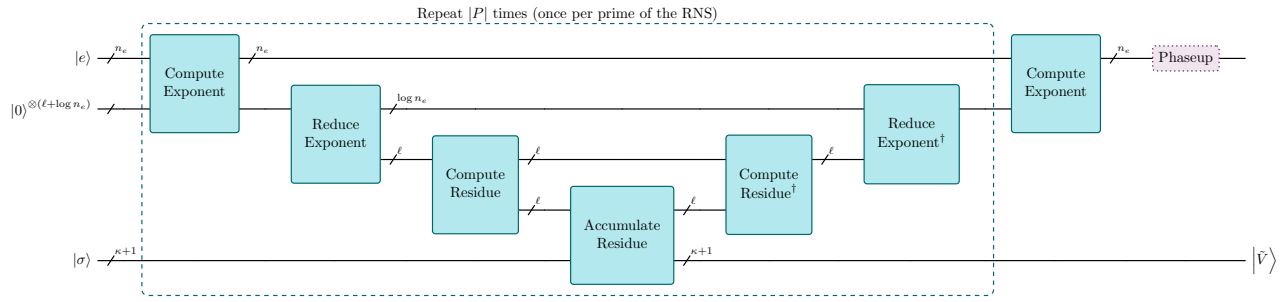


Figure 26: Overview of the oracle of Shor's algorithm at the logical level. It shifts the accumulator, prepared in the mask state $|\sigma\rangle$, by $\tilde{V}(e) = \sum_j \sum_k r_{j,k} C_{j,k} \bmod (N \gg \bar{\kappa})$, the top κ bits of $g^e \bmod N$ up to the truncation error (Equation (52)). In iteration j, for the prime p_j, ComputeExponent forms $S_j = \sum_k e_k D_{p_j,k}$, ReduceExponent replaces it by $S_j \bmod (p_j - 1)$ and a quotient register, ComputeResidue forms $r_j = g_{p_j}^{S_j \bmod (p_j-1)} \bmod p_j$, and AccumulateResidue adds $\sum_k r_{j,k} C_{j,k}$. The two adjoints then erase r_j and restore S_j. After the loop, a last ComputeExponent erases $S_{|P|-1}$, and a Phaseup applies the deferred phase corrections on $|e\rangle$. The subroutines are detailed in Section B.1 and listed in order in Algorithm 1. This is the analog of Figure 13b, restricted to the oracle.

We now detail each of the subroutines, starting with the computation of the logarithm of the residue.

B.1.1 Computing the logarithm of the residue

The goal of ComputeExponent is, given the classically known discrete logarithms $D_{p_i,k}$, and the exponent e, to compute S_i which is defined as

$$S_i = \sum_{k=0}^{n_e-1} D_{p_i,k} e_k, \tag{64}$$

Since,

$$r_i = g_{p_i}^{\sum_{k=0}^{n_e-1} D_{p_i,k} e_k} \mod p_i = g_{p_i}^{S_i} \mod p_i, \tag{65}$$

each S_i is the exponent used to compute the residue r_i. Following the implementation of [11], we apply three optimizations for the computation of S_i.

First, as explained in Appendix A, windowing is used to reduce the number of iterations on the oracle's exponent e from n_e to $\lceil \frac{n_e}{w_1} \rceil$. Each window of w_1 qubits is the address register of a Lookup and adds a $2^{w_1} - w_1 - 1$ Toffoli gate cost.

Second, the computation of S_i can be merged with the uncomputation of S_{i-1}. Indeed, since the addition that computes S_i and the subtraction that uncomputes S_{i-1} depend on the same address register and windowed exponent $e_{k:k+w_1}$, the two operations can be merged into the same loading operation. This means S_i can be expressed as follows,

$$S_i = S_{i-1} + \sum_{j=0}^{\lceil \frac{n_e}{w_1} \rceil - 1} \mathrm{T}_1^{(i,j)}[e_{jw_1:(j+1)w_1}], \tag{66}$$

where

$$\mathrm{T}_1^{(i,j)}[e_{jw_1:(j+1)w_1}] \coloneqq \sum_{k=0}^{w_1-1} (D_{p_i,jw_1+k} - D_{p_{i-1},jw_1+k}) \times e_{jw_1+k} \tag{67}$$

is the value loaded by the Lookup operation.

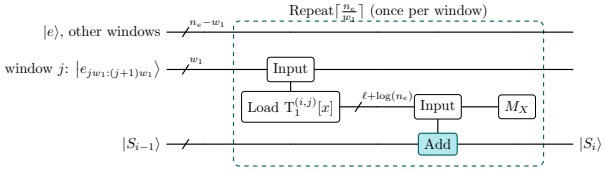


Figure 27: ComputeExponent: one pass of the loop that turns S_{i-1} into S_i, where $S_i = \sum_k e_k D_{p_i,k}$ is held on $\ell + \log n_e$ qubits and $D_{p_i,k}$ is the classically precomputed discrete logarithm of $M_k = g^{2^k} \bmod N$ modulo p_i (Table 8). The loop runs over the $\lceil n_e/w_1 \rceil$ windows of the exponent. In iteration j, window j is the address x of a Lookup, which loads $\mathrm{T}_1^{(i,j)}[x] = \sum_{k=0}^{w_1-1} x_k \left(D_{p_i,jw_1+k} - D_{p_{i-1},jw_1+k}\right)$. The loaded value is added into S and then measured, and the other windows are untouched. Adding these differences uncomputes S_{i-1} and computes S_i in the same pass.

The third optimization is to delay and merge Phaseups associated to measurement-based uncomputation of the Lookups. Once the value loaded into a register by the Lookup operation has been used, this register is measured in the X basis. Usually, a Phaseup operation would need to be applied immediately to apply phase corrections to the address register. However, in this context, the address register that contains the exponent is only used as control throughout the entire computation. Thus every phase correction on the exponent register can be pushed to the end of the computation and combined into a single Phaseup operation for each window of w_1 qubits. The overall circuit for ComputeExponent with the three optimizations is depicted in Figure 27. Its total cost is given in Table 9.

B.1.2 Reducing the exponent

ComputeExponent leaves S_i on $\ell + \log(n_e)$ qubits, but ComputeResidue only needs it modulo $p_i - 1$,

Calls	Operation	Operation size
$(\lvert P\rvert+1)\lceil\frac{n_e}{w_1}\rceil$	Q-Q Adder	$\ell+\log(n_e)$
	Lookup	address $= w_1$, target $= \ell+\log(n_e)$
$\lceil\frac{n_e}{w_1}\rceil$	Phaseup	address $= w_1$

Table 9: Enumeration of operations used by ComputeExponent. The Phaseups of all Lookups are deferred to the end of the computation and merged into one Phaseup per window of w_1 exponent qubits (last row).

the order of the generator g_{p_i}. ReduceExponent therefore replaces S_i by its remainder $S_i \bmod (p_i-1)$, which lies in $[0, p_i-1)$ and fits on ℓ qubits. This shortens the exponent that ComputeResidue has to process, and so lowers its cost. The reduction has to be reversible, because ReduceExponent† must later restore S_i. ReduceExponent is therefore a reversible Euclidean division of S_i by p_i-1. It keeps the remainder in the ℓ least significant qubits of the register, and the quotient in the $\log(n_e)$ most significant ones.

i Long division, one bit at a time. ReduceExponent follows schoolbook long division in binary. An $(\ell+1)$-qubit window moves down the register of S_i, one position per step, starting from the most significant end. At each step the window holds twice the current remainder plus the next bit of S_i. A single conditional subtraction of p_i-1 brings it back below p_i-1, and the window then moves down by one position. After $\log(n_e)$ steps the window has reached the bottom of the register, and its ℓ low qubits hold $S_i \bmod (p_i-1)$, as shown in Figure 28.

Each step is valid because its input stays below $2(p_i-1)$. The discrete logarithms satisfy $D_{p_i,k} \le p_i-2$, so $S_i \le n_e(p_i-2) < 2^{\log(n_e)}(p_i-1)$, with $\log(n_e) = \lceil \log_2 n_e \rceil$. The ℓ most significant qubits of S_i therefore already hold a value below p_i-1. By induction, each window holds twice a remainder below p_i-1 plus one bit, so its value is below $2(p_i-1)$.

ii One reduction step. A step must reduce an input x with $0 \le x < 2b$, where $b = p_i-1$, by subtracting b at most once. Lemma B.1 does this without any controlled adder.

Lemma B.1. *Let b be an n-bit integer, $2^{n-1} \le b \le 2^n-1$, and let $0 \le x \le 2b-1$ be stored on $n+1$ qubits. The following sequence uses an n-qubit auxiliary register initialized to $|0\rangle$. It leaves $x \bmod b$ in the n low qubits of the working register and the bit $c = \mathbb{1}_{x<b}$ in its top qubit, it returns the auxiliary register to $|0\rangle$.*

$$\begin{aligned}
|x\rangle &\xmapsto{\text{Q-C Adder}^\dagger} |x-b\rangle \\
&\xmapsto{CX(n)} |c\rangle\,|y_t = x-b+2^n c\rangle\,|0 \oplus bc\rangle \\
&\xmapsto{\text{Q-Q Adder}} |c\rangle\,|y_t+bc\rangle\,|bc\rangle \\
&\xmapsto{CX(n)} |c\rangle\,|y_t+(b-2^n)c\rangle\,,
\end{aligned} \tag{68}$$

where $c = \mathbb{1}_{x<b} = 1$ if $x < b$, and 0 otherwise.

Proof. Since $0 \le x \le 2b-1$, there are exactly two mutually exclusive cases, either $0 \le x < b$, or $b \le x \le 2b-1$.
If $x < b$: then $\mathbb{1}_{x<b} = 1$, and the working register contains $x-b+2^n+(b-2^n) = x = x \bmod b$.
If $x \ge b$: then $\mathbb{1}_{x<b} = 0$, and the working register contains $x-b = x \bmod b$. □

Here $x-b$ is stored in two's complement on $n+1$ qubits, so $|x-b\rangle = |\mathbb{1}_{x<b}\rangle\,|y_t\rangle$. Its top qubit is the sign bit, and its n low qubits hold y_t.

The direct implementation would first compute $\mathbb{1}_{x<b}$ with a Q-C Comparison, then use it to control a Q-C Adder† that subtracts b. That controlled Q-C Adder† costs $2n+O(1)$ Toffoli gates. The lemma avoids it. It always subtracts b, with an uncontrolled Q-C Adder† whose top qubit records the underflow. The underflow qubit then controls a multi-target Cnot gate ($CX(n)$), which loads b into the auxiliary register if $x < b$, and 0 otherwise. A Q-Q Adder adds the auxiliary register back, which undoes the subtraction exactly when $x < b$. A second $CX(n)$ clears the auxiliary register. This saves n Toffoli gates, at the cost of the n-qubit auxiliary register, which is free at this point of the circuit anyway. The same construction, with small variations, implements the modular additions of ComputeResidue and AccumulateResidue.

iii The quotient register. In each step, the top qubit of the window ends in the state $|\mathbb{1}_{x<b}\rangle$ and is not measured. It then leaves the window, while the next lower qubit of S_i enters it. We call the $\log(n_e)$ qubits produced in this way the *quotient qubits*, and the register they form the *quotient register*. The quotient qubit produced at step k holds the complement of bit $\log(n_e)-k-1$ of the quotient a, where $S_i \bmod (p_i-1) = S_i - a(p_i-1)$. The quotient register therefore stores a bitwise complemented, which is exactly the information the reversible division must retain. It occupies the $\log(n_e)$ most significant qubits of the $(\ell+\log(n_e))$-qubit input register, the ℓ least significant ones holding $S_i \bmod (p_i-1)$. It stays allocated during ComputeResidue, AccumulateResidue and ComputeResidue†, until ReduceExponent† runs the same steps in reverse and uses it to restore S_i. All additions and subtrac-

tions in ReduceExponent act on registers of the same size. Their number is given in Table 10.

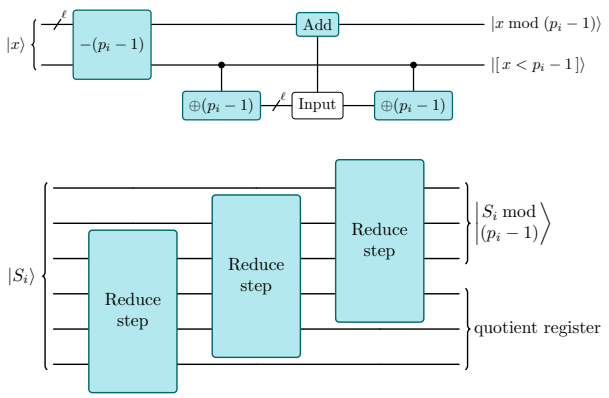


Figure 28: ReduceExponent. Top: one reduction step, the construction of Lemma B.1 with $b = p_i - 1$. For an input $0 \le x < 2(p_i - 1)$ on $\ell + 1$ qubits, it leaves $x \bmod (p_i - 1)$ on the low ℓ qubits and the bit $[x < p_i - 1]$, the $\mathbb{1}_{x<b}$ of the lemma, on the top qubit. Bottom: ReduceExponent for $\ell = \log n_e = 3$, as $\log n_e$ such steps, each on an $(\ell+1)$-qubit window that moves down by one qubit (Section B.1.2). The top qubits left behind by the steps form the quotient register, which is kept to uncompute ReduceExponent.

Calls	Operation	Operation size
$\lvert P\rvert \log(n_e)$	Q-Q Adder	ℓ
	Q-C Adder (subtraction)	$\ell + 1$

Table 10: Enumeration of operations used for ReduceExponent.

B.1.3 Computing the residue and its uncomputation

ComputeResidue turns the reduced exponent into the residue $r_i = g_{p_i}^{S_i \bmod (p_i-1)} \bmod p_i$. It is a small modular exponentiation on ℓ-qubit registers, built from windowed table lookups and modular additions in the style of Lemma B.1. Its adjoint ComputeResidue† is not obtained by running ComputeResidue backwards. ComputeResidue measures its auxiliary registers in the X basis and defers the resulting phase corrections, so ComputeResidue† is a circuit of its own. It erases r_i and applies all the deferred corrections at once. We first describe ComputeResidue, then ComputeResidue†, and finally the phase corrections they share.

i ComputeResidue. ComputeResidue takes as input the reduced exponent $S_i \bmod (p_i - 1)$, which we write S_i for the rest of this subsection, and outputs

$$r_i = g_{p_i}^{S_i} \bmod p_i. \tag{69}$$

Following [63], it splits S_i into $\lceil \ell / w_{3a} \rceil$ windows of w_{3a} bits, with offsets $J_{j_a} = j_a w_{3a}$, and writes

$$r_i = \prod_{j_a=0}^{\lceil \ell/w_{3a} \rceil - 1} g_{p_i}^{S_{i,J_{j_a}:J_{j_a+1}} 2^{J_{j_a}}} \bmod p_i. \tag{70}$$

Let $\alpha_{j_a} = g_{p_i}^{S_{i,0:J_{j_a}}} \bmod p_i$ be the partial product of the first j_a windows. Each factor multiplies the partial product by a constant that depends on one window of S_i, so that $\alpha_{j_a+1} = \alpha_{j_a} \, g_{p_i}^{S_{i,J_{j_a}:J_{j_a+1}} 2^{J_{j_a}}} \bmod p_i$.

This multiplication is itself windowed. The partial product α_{j_a} is split into $\lceil \ell / w_{3b} \rceil$ windows of w_{3b} bits, with offsets $J_{j_b} = j_b w_{3b}$. Write $x_a := S_{i,J_{j_a}:J_{j_a+1}} \in \{0, \dots, 2^{w_{3a}} - 1\}$ for the current window of the exponent, and $y_b := \alpha_{j_a, J_{j_b}:J_{j_b+1}} \in \{0, \dots, 2^{w_{3b}} - 1\}$ for a window of the partial product. The multiplication then becomes a sum of table entries,

$$\alpha_{j_a+1} = \sum_{j_b=0}^{\lceil \ell/w_{3b} \rceil - 1} \mathrm{T}_3^{(i,j_a,j_b)}[x_a, y_b] \bmod p_i,$$
$$\mathrm{T}_3^{(i,j_a,j_b)}[x_a, y_b] := y_b \, 2^{J_{j_b}} \, g_{p_i}^{x_a 2^{J_{j_a}}} \bmod p_i. \tag{71}$$

The table is precomputed classically for every address pair (x_a, y_b). The inner loop over j_b therefore computes α_{j_a+1} out of place, with one Lookup and one modular addition per window. The outer loop over j_a chains these multiplications into the exponentiation.

ii The modular addition. Each step of the inner loop adds a looked-up value to the helper register modulo p_i, with a variant of Lemma B.1. Write T for $\mathrm{T}_3^{(i,j_a,j_b)}[x_a, y_b]$. The Lookup loads $p_i - T$ rather than T, and a Q-Q Adder† replaces the Q-C Adder† of the lemma,

$$|p_i - T\rangle \, |h\rangle \longmapsto |p_i - T\rangle \, |h - (p_i - T)\rangle \, .$$

This single subtraction both adds T and performs the subtraction of p_i that the modular reduction needs. When it underflows, the last three steps of Equation (68) add p_i back. The loaded register is then measured in the X basis, and its phase correction is deferred to ComputeResidue†.

iii Initialization. The outer loop starts from $\alpha_2 = g_{p_i}^{S_{i,0:J_2}} \bmod p_i$, loaded directly by a Lookup of address size $2w_{3a}$. This skips the first two multiplications, by $g_{p_i}^{S_{i,0:J_1}}$ and by $g_{p_i}^{S_{i,J_1:J_2} 2^{J_1}}$. When $w_{3a} = w_{3b}$, as in our resource estimates, this Lookup has the same address size as those of the inner loop, namely $w_{3a} + w_{3b}$. One iteration of the outer loop then reads

$$\begin{aligned} |S_i\rangle \, |\alpha_{j_a}\rangle \, |0\rangle^{\otimes \ell} &\xmapsto{\text{mod mul}} |S_i\rangle \, |\alpha_{j_a}\rangle \, |\alpha_{j_a+1}\rangle \\ &\xmapsto{\text{role swap}+M_X} |S_i\rangle \, |\alpha_{j_a+1}\rangle \, . \end{aligned} \tag{72}$$

Figure 29 shows the circuit, and Table 11 counts its operations.

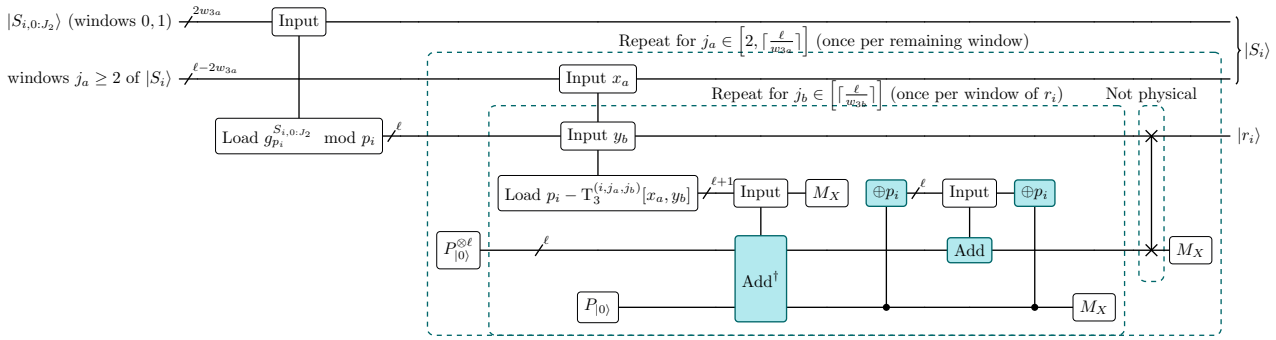


Figure 29: ComputeResidue, a windowed modular exponentiation $r_i = g_{p_i}^{S_i \bmod (p_i-1)} \bmod p_i$, built like the exponentiation of Figure 8. A first Lookup loads the product of the first two windows, $g_{p_i}^{S_{i,0:J_2}} \bmod p_i$. Each outer iteration j_a then multiplies the partial product by the power of g_{p_i} selected by x_a, window j_a of S_i. The multiplication is out of place, each inner iteration j_b adds the table entry $\mathrm{T}_3^{(i,j_a,j_b)}[x_a, y_b]$, where y_b is window j_b of the partial product, with the modular addition of Lemma B.1. The swap at the end of each outer iteration only exchanges the roles of the two registers, in software. The phase corrections of the measurements, including the UnderflowPhasefix, are deferred to ComputeResidue† (Figure 30).

Calls	Operation	Operation size
$\lvert P\rvert$	Lookup (initialization)	address $= 2w_{3a}$, target $= \ell$
$\lvert P\rvert \left(\lceil \frac{\ell}{w_{3a}} \rceil - 2\right) \lceil \frac{\ell}{w_{3b}} \rceil$	Q-Q Adder (subtraction)	$\ell + 1$
	Q-Q Adder	ℓ
	Lookup	address $= w_{3a} + w_{3b}$, target $= \ell + 1$

Table 11: Operations used by ComputeResidue. For resource estimation, we choose $w_{3a} = w_{3b} = w_3$.

iv ComputeResidue†. ComputeResidue† erases the residue,

$$\text{ComputeResidue}^\dagger \, |S_i\rangle \, |r_i\rangle = |S_i\rangle \, |0\rangle \,, \tag{73}$$

and applies all the phase corrections that ComputeResidue deferred. It runs the outer loop backwards. Each iteration turns α_{j_a+1} back into α_{j_a} with two inner loops, built from the same step as the inner loop of ComputeResidue (Figure 30). The first inner loop computes α_{j_a} out of place, from α_{j_a+1}, with the table of inverse factors

$$T_{3,\text{inv}}^{(i,j_a,j_b)}[x_a, y_b] := y_b \, 2^{J_{j_b}} \, g_{p_i}^{-x_a 2^{J_{j_a}}} \bmod p_i .$$

The second inner loop uses α_{j_a} to clear the register that held α_{j_a+1}. It loads $T_3^{(i,j_a,j_b)}[x_a, y_b]$ instead of $p_i - T_3^{(i,j_a,j_b)}[x_a, y_b]$, so that each pass subtracts the table entry modulo p_i instead of adding it. It runs over the windows j_b in reverse order, so that each of its passes undoes one pass of the inner loop of ComputeResidue, from the last to the first. One iteration then reads

$$\begin{aligned} |S_i\rangle \, |\alpha_{j_a+1}\rangle &\xmapsto{\text{innerloop 1}} |S_i\rangle \, |\alpha_{j_a+1}\rangle \, |\alpha_{j_a}\rangle \\ &\xmapsto{\text{innerloop 2}} |S_i\rangle \left| \alpha_{j_a+1} - \alpha_{j_a} \, g_{p_i}^{x_a 2^{J_{j_a}}} \right\rangle |\alpha_{j_a}\rangle \\ &= |S_i\rangle \, |0\rangle \, |\alpha_{j_a}\rangle \,, \end{aligned} \tag{74}$$

since $\alpha_{j_a} = \alpha_{j_a+1} \, g_{p_i}^{-x_a 2^{J_{j_a}}} \bmod p_i$. A role swap then makes α_{j_a} the input of the next iteration. After the last iteration, the register holds α_2, which is unloaded by measurement-based uncomputation and a Phaseup addressed by the $2w_{3a}$ low bits of S_i, mirroring the initialization of ComputeResidue. Equation (73) then holds. Table 12 counts the operations.

v Phase corrections. Each step measures two kinds of qubits in the X basis: the register loaded by the Lookup, and the underflow qubit of the modular addition. Both measurements leave phases that must be corrected.

The phases left by the measurement of the loaded registers are corrected by Phaseup operations. Indeed, a loaded register measured with outcomes m leaves a phase $(-1)^{m \cdot T[x]}$, which depends only on the address x. To correct this added phase, all corrections that share an address are summed classically and applied with a single Phaseup. The Phaseup of the first inner loop of ComputeResidue† corrects the register loaded in this loop and the register loaded by the Lookup inside the UnderflowPhasefix of this loop. The Phaseup of the second inner loop is addressed by x_a and a window y_b of α_{j_a}, which is the same address as the pass j_b of iteration j_a of ComputeResidue. Since the same address is retrieved, the phases left by ComputeResidue can be corrected here. The Phaseup of the second inner loop thus corrects three registers at once, the register loaded by that pass of ComputeResidue and by the second inner loop, and the register loaded inside the Under-

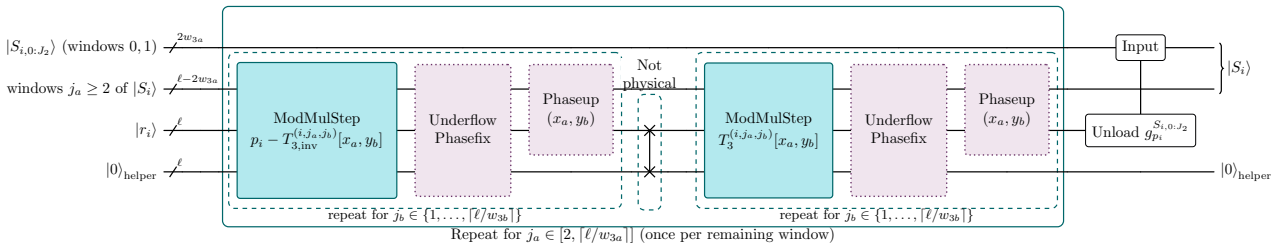


Figure 30: ComputeResidue†. It erases the residue r_i computed by ComputeResidue (Figure 29) and applies the phase corrections that ComputeResidue deferred. It runs the outer loop of Figure 29 again, over the windows x_a of S_i. Each ModMulStep box is one pass of the inner loop there, that is, one step of the out-of-place multiplication shown at the bottom of Figure 8. It loads $p_i - T^{(i,j_a,j_b)}_{3,\text{inv}}[x_a, y_b]$ in the first inner loop and $T^{(i,j_a,j_b)}_{3}[x_a, y_b]$ in the second, which subtracts that entry modulo p_i, where y_b is window j_b of the partial product. The second inner loop runs over j_b in reverse order, so that each of its passes undoes one pass of the inner loop of ComputeResidue. Each pass is followed by the underflow phase correction of Figure 31 and by a Phaseup addressed by (x_a, y_b). In the second inner loop, the underflow phase correction compares with $p_i - T^{(i,j_a,j_b)}_{3}[x_a, y_b]$ and also corrects the phase of the matching pass of ComputeResidue. A Phaseup applies the phase $(-1)^{m \cdot T[x]}$ to the address x of a Lookup whose output $T[x]$ was measured in the X basis with outcomes m, which completes its measurement-based uncomputation [77]. The swap between the two inner loops only exchanges the roles of the two registers, in software. After the outer loop, the initial value $g_{p_i}^{S_{i,0:J_2}}$ is unloaded by measurement-based uncomputation, with a Phaseup addressed by the first two windows of S_i.

ComputeResidue†		
Calls	Operation	Operation size
$\lvert P\rvert$	Phaseup (final unloading)	address $= 2w_{3a}$
$2\lvert P\rvert \left(\lceil \frac{\ell}{w_{3a}} \rceil - 2\right) \lceil \frac{\ell}{w_{3b}} \rceil$	Q-Q Adder (subtraction)	$\ell + 1$
	Q-Q Adder	ℓ
	Lookup	address $= w_{3a} + w_{3b}$, target $= \ell + 1$
	Phaseup	$w_{3a} + w_{3b}$
$\lvert P\rvert \left(\lceil \frac{\ell}{w_{3a}} \rceil - 2\right) \lceil \frac{\ell}{w_{3b}} \rceil$	UnderflowPhasefix	address $= w_{3a} + w_{3b}$, comparison $= \ell$
UnderflowPhasefix		
Calls	Operation	Operation size
$\lvert P\rvert \left(\lceil \frac{\ell}{w_{3a}} \rceil - 2\right) \lceil \frac{\ell}{w_{3b}} \rceil$	Lookup	address $= w_{3a} + w_{3b}$, target $= \ell$
	Q-Q Comparison	ℓ

Table 12: Operations used by ComputeResidue† and by UnderflowPhasefix. For resource estimation, we choose $w_{3a} = w_{3b} = w_3$.

flowPhasefix of this loop.

The phase left by the measurement of the underflow qubit is corrected by UnderflowPhasefix, which needs the value of that qubit. Let x be the input of the step. It is either 0 or the output of the previous step, so $0 \leq x < p$, with $p = p_i$. Let $p - T$ be the loaded value, with $0 \leq T < p$, such that $0 < p - T \leq p$. The Lookup loads p, not 0, when $T = 0$, which happens whenever $y_b = 0$. The underflow qubit u is the sign bit of

$$x - (p - T) = x + T - p. \tag{75}$$

Computing the comparison $u = \mathbb{1}_{x<p-T}$ directly is not possible, because x is gone. The helper register now holds $x + T$ if $u = 1$, and $x + T - p$ if $u = 0$. Since $x \geq 0$, the first case gives $x + T \geq T$, and since $x < p$, the second gives $x + T - p < T$. Comparing the helper register with T therefore recovers u, as $u = \mathbb{1}_{y \geq T}$ for the output y of the step. The second inner loop of ComputeResidue† loads and subtracts T instead. The same argument, with T and $p - T$ exchanged, gives $u = \mathbb{1}_{y \geq p-T}$. In both cases, the underflow qubit is recovered by comparing the output with p minus the loaded value. UnderflowPhasefix (Figure 31) loads that value, performs this comparison, and corrects the phase if needed. Unlike the other phase corrections, UnderflowPhasefix cannot be merged into a single Phaseup, because the correction depends on the helper register, which changes at every pass. It must be applied at a pass where that register holds the value that determines u.

One UnderflowPhasefix per pair of passes can be saved, as in [11]. Pass j_b of the second inner loop of ComputeResidue† undoes pass j_b of iteration j_a of ComputeResidue. Both passes have the address (x_a, y_b) and the table entry $T = \mathrm{T}^{(i,j_a,j_b)}_{3}[x_a, y_b]$. The pass of ComputeResidue loads $p_i - T$ and maps the helper register from some value h to $h + T \bmod p_i$. Since the second inner loop runs in reverse order, the

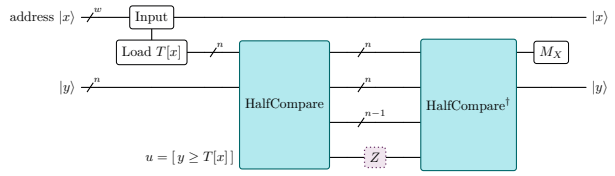


Figure 31: UnderflowPhasefix, the phase correction $|x\rangle\,|y\rangle \mapsto (-1)^{[\,y\geq T[x]\,]}\,|x\rangle\,|y\rangle$, applied when the X-basis measurement of the underflow qubit of a modular addition $y+T[x]$ returns 1. A Lookup loads $T[x]$. HalfCompare is the first half of the subtraction $y-T[x]$, whose last carry is the comparison bit $u=[\,y\geq T[x]\,]$. A Z gate on u applies the phase, and HalfCompare† erases the carries. The loaded value is then measured in the X basis. Its phase correction acts on the same address x, so it is merged into the Phaseup that follows in Figure 30.

pass of ComputeResidue† starts from $h+T \bmod p_i$, loads T, and returns the register to h. The two underflow qubits are respectively $u_c = \mathbb{1}_{h<p_i-T}$ and $u_2 = \mathbb{1}_{(h+T \bmod p_i)<T}$. If $u_c = 1$, no reduction occurs, hence $h+T \bmod p_i \geq T$, and thus $u_2 = 0$. If $u_c = 0$, then $h+T-p_i < T$, and $u_2 = 1$. The two qubits are therefore complementary, $u_2 = 1-u_c$. Let m_c and m_{u2} denote the X-basis outcomes of u_c and u_2. The two measurements leave the phase

$$(-1)^{m_c u_c + m_{u2} u_2} = (-1)^{m_{u2}}\,(-1)^{(m_c \oplus m_{u2})\,u_c}. \quad (76)$$

The first factor is a global phase. The second factor is corrected by a single UnderflowPhasefix, applied only if $m_c \oplus m_{u2} = 1$, which happens with probability $1/2$. The UnderflowPhasefix acts right after the pass of ComputeResidue†, when the register holds h again, and compares h with $p_i - T$, which gives u_c. Note that if ComputeResidue loaded 0 for $T=0$, both underflow qubits would be 0 on that branch, and the global phase $(-1)^{m_{u2}}$ would become a relative phase, hence the convention $0 < p_i - T \leq p_i$. In addition, the second inner loop must run in the reverse order. In the forward order, the helper register never holds the values h of ComputeResidue again, so the underflow phases of ComputeResidue cannot be corrected.

B.1.4 Accumulating the residue

AccumulateResidue adds the contribution of the residue r_i to the accumulator, so that after the last prime the accumulator holds $\tilde{V}$ (Equation (52)). It is the only subroutine that writes to the accumulator. It reuses the modular addition of ComputeResidue, on a $(\kappa+1)$-qubit register instead of an $(\ell+1)$-qubit one.

The subroutine takes as input the current residue $|r_i\rangle$ and the accumulated value $|\mathrm{Acc}_{i-1}\rangle$, with $\mathrm{Acc}_i = \sum_{j=0}^{i}\sum_{k=0}^{\ell-1} r_{j,k}C_{j,k}$ the partial sum for the i first primes of the residue number system. It performs an addition modulo $(N \gg \bar{\kappa})$ and outputs $|r_i\rangle$ and $|\mathrm{Acc}_i\rangle$. Recall that $(N \gg \bar{\kappa}) = \lfloor \frac{N}{2^{\bar{\kappa}}} \rfloor$ is obtained from the unsigned integer binary representation of N by deleting its $\bar{\kappa}$ least significant bits. The circuit is presented in Figure 32, and is very similar to one inner loop of ComputeResidue† detailed before in Figure 30, albeit using different tables and operations on registers of different sizes. The number of operations required for the accumulation is given in Table 13.

B.1.5 Logical cost breakdown

This subsection collects the operation and qubit counts of all subroutines, for use in the resource estimate of Appendix C. For additions, the size given is the size of the input number.

For Lookup operations, the values given are the address register and target register sizes. For Phaseup operations, the value given is the address register size, which is the only value needed as the target register is only measured out in the X basis before starting the Phaseup operation. For Comparison operations, the circuit is a subtraction and uses the last carry bit as the comparison bit, thus a comparison of size n is equivalent in cost to an addition of size $n+1$. Additions, subtractions and comparisons are referred to and accounted for as Adder operations as their circuit and costs are similar. Concerning the qubit cost, the logical qubit count at the different steps of the circuit is shown in Table 15. Observe that the total cost consists mostly of Adder operations, Lookup operations and, to a lesser extent, Phaseup operations. Therefore, it is important to optimize their cost and implementation at the physical level, which is the focus of Appendix C.

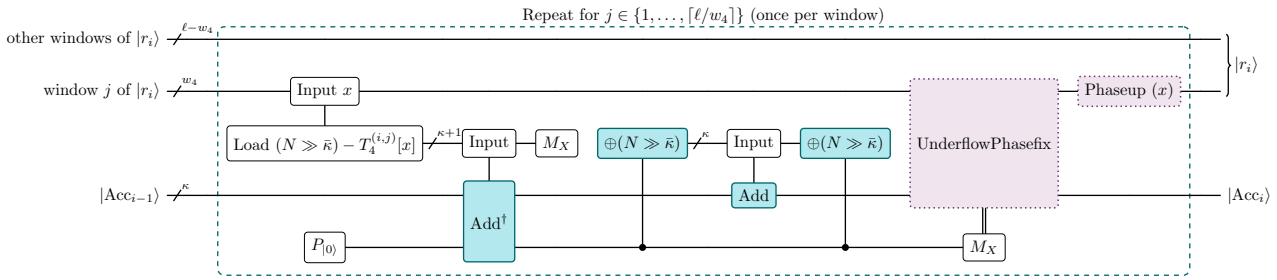


Figure 32: AccumulateResidue, the multiply-accumulate $\mathrm{Acc}_i = \mathrm{Acc}_{i-1} + \sum_k r_{i,k} C_{i,k} \bmod (N \gg \bar{\kappa})$, with the constants $C_{i,k}$ of Equation (52). It multiplies the bit vector of r_i by classical constants, window by window. In iteration j, window j of r_i is the address x of a Lookup that loads $(N \gg \bar{\kappa}) - T_4^{(i,j)}[x]$, with $T_4^{(i,j)}[x] = \sum_{k=0}^{w_4-1} x_k C_{i,jw_4+k}$. Subtracting this value adds $T_4^{(i,j)}[x]$ modulo $N \gg \bar{\kappa}$. The loaded value is then measured in the X basis, and the Phaseup addressed by x applies the matching phase correction, as in Figure 30. UnderflowPhasefix (Figure 31) corrects the phase left by measuring the underflow qubit of the modular addition.

AccumulateResidue		
Calls	Operation	Operation size
$\lvert P\rvert\lceil \frac{\ell}{w_4} \rceil$	Q-Q Adder (subtraction)	$\kappa + 1$
	Q-Q Adder	κ
	Lookup	address $= w_4$, target $= \kappa + 1$
	Phaseup	w_4
$\frac{\lvert P\rvert}{2}\lceil \frac{\ell}{w_4} \rceil$	UnderflowPhasefix	address $= w_4$, comparison $= \kappa$
UnderflowPhasefix		
Calls	Operation	Operation size
$\frac{\lvert P\rvert}{2}\lceil \frac{\ell}{w_4} \rceil$	Lookup	address $= w_4$, target $= \kappa$
	Q-Q Comparison	κ

Table 13: Enumeration of operations used by AccumulateResidue and the UnderflowPhasefix.

B.2 Arithmetic circuits for the two ECDLP variants

B.2.1 High-level point addition

Recall that elliptic curve points are represented in affine coordinates (x, y) over the prime field $\mathbb{F}_q$. This representation misses the point at infinity, but with very high probability the operations will not produce it, so that the point addition remains correct for most cases. Likewise, the doubling case is omitted from the addition formulas.

Addition by a constant point (x_2, y_2) is thus performed by Algorithm 2, like in [4, 71].

In this algorithm, the constant (x_2, y_2) is not hardcoded into the circuit: it is loaded in memory using three Lookups. The first one loads (x_2, y_2) at the beginning of the algorithm, the last one loads (x_2, y_2) at the end, and the middle one loads $3x_2$ only. In the case of [6, 12], there are enough ancilla qubits available at the time of the Lookups to neglect their effect on the total qubit count. This is not the case in [13], but in the latter, one can look up each coordinate individually instead with only a minor impact on the total gate count.

The implementation of modular subtraction and negation also has a minor impact on the gate count, so in the following we focus on in-place modular multiplication $(x_1, y_1 \leftarrow x_1, y_1 \cdot x_1)$ and out-of-place squaring $(x_1 \leftarrow x_1 - y_1^2)$. Both use the same ancilla qubits, as shown in Figure 35.

Algorithm 2 Constant point addition on an ordinary prime-field curve, in affine coordinates.

Input: Registers x_1, y_1 (coordinates of the accumulator point)
Output: Registers are modified and contain the sum of (x_2, y_2) with (x_1, y_1)

1: $x_1 \leftarrow x_1 - x_2$
2: $y_1 \leftarrow y_1 - y_2$
3: $x_1, y_1 \leftarrow x_1, y_1 \cdot x_1^{-1}$
4: $x_1 \leftarrow x_1 + 3x_2$
5: $x_1 \leftarrow x_1 - y_1^2$
6: $x_1, y_1 \leftarrow x_1, y_1 \cdot x_1$
7: $x_1 \leftarrow -x_1$
8: $y_1 \leftarrow y_1 - y_2$
9: $x_1 \leftarrow x_1 + x_2$

Subroutine	Calls	Register size	Address size	Q-Q Adder	Q-C Adder	Lookup	Phaseup
ComputeExponent	$(\lvert P\rvert+1)\lceil\frac{n_e}{w_1}\rceil$	$\ell+\log(n_e)$	w_1	1	0	1	0
ReduceExponent	$\lvert P\rvert\log(n_e)$	$\ell+1$	-	1	1	0	0
ComputeResidue (initialization)	$\lvert P\rvert$	ℓ	$2w_3$	0	0	1	0
ComputeResidue (outer loop)	$\lvert P\rvert\left(\lceil\frac{\ell}{w_3}\rceil-2\right)\lceil\frac{\ell}{w_3}\rceil$	$\ell+1$	$2w_3$	2	0	1	0
AccumulateResidue	$\lvert P\rvert\lceil\frac{\ell}{w_4}\rceil$	$\kappa+1$	w_4	2.5	0	1.5	1
ComputeResidue† (outer loop)	$2\lvert P\rvert\left(\lceil\frac{\ell}{w_3}\rceil-2\right)\lceil\frac{\ell}{w_3}\rceil$	$\ell+1$	$2w_3$	2.5	0	1.5	1
ComputeResidue† (unloading)	$\lvert P\rvert$	ℓ	$2w_3$	0	0	0	1
ReduceExponent†	$\lvert P\rvert\log(n_e)$	$\ell+1$	-	1	1	0	0
Last Phaseup (ComputeExponent)	$\lceil\frac{n_e}{w_1}\rceil$	-	w_1	0	0	0	1

Table 14: Enumeration of operations used by each subroutine of the algorithm. "Register size" refers to the biggest register involved in an addition or as the target of a Lookup in that subroutine. Comparisons are counted as Q-Q Adders. The 0.5 comes from the UnderflowPhasefix which is applied or not 50 % of the time (depends on an M_X measurement). The last row gives the $\lceil\frac{n_e}{w_1}\rceil$ Phaseups that apply, at the end of the computation, the phase corrections deferred by ComputeExponent. These Phaseups act on no Adder or Lookup target, hence the empty register size.

Algorithm 3 Dialog computation.

Input: x
Output: Dialog vector **g**
$(u, v) := (q, x)$
for i in range(N_{it}) **do**
  $b_0 := v \bmod 2$
  $b_1 := u > v$
  Store $b_0 \& b_1$ and b_0 in **g**
  if $b_0 \& b_1$ **then**
    $u, v = v, u$
  end if
  if b_0 **then**
    $v -= u$ ▷ Integer subtraction
  end if
  $v = v/2$ ▷ Built-in by swapping qubit indices
end for ▷ At the end, $v = 0$ and $u = 1$

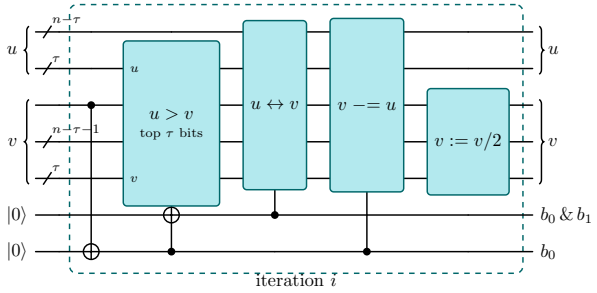


Figure 33: One iteration i of the dialog construction, pass 1 of Algorithm 3.

Algorithm 4 Replay of the dialog

Input: Integer y, dialog **g**
Output: Replaces y by $yx \bmod q$
$r, s \leftarrow y, 0$
for i in reversed(range(N_{it})) **do**
  Obtain b_0 and $b_0 \& b_1$ for iteration i from **g**
  $s \leftarrow 2s \bmod q$
  if b_0 **then**
    $s \leftarrow s + r \bmod q$
  end if
  if $b_0 \& b_1$ **then**
    $r, s \leftarrow s, r$
  end if
end for
$(r, s) = (0, yx \bmod q)$

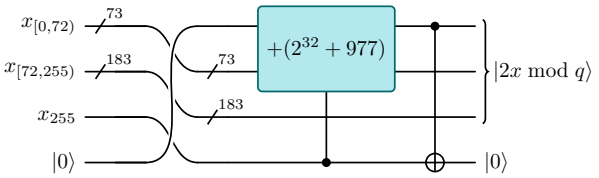


Figure 34: Optimized modular doubling for $q = 2^{256} - (2^{32} + 977)$ with 40 bits of padding.

B.2.2 Out-of-place squaring

Following [12], out-of-place squaring is implemented by a sequence of modular doubling and modular adders.

B.2.3 In-place modular inverse using dialogs

Modular inverse using dialogs is a desynchronized variant of the binary extended GCD. A fixed number of iterations $N_{it} = 1.413n + O(\sqrt{n})$ is selected, where the choice of the constant in the O determines the success probability of the algorithm. On input (x, y), the algorithm runs in three phases, which are implemented by three circuits: 1. computation of the dialog (Algorithm 3, transforms x in-place into the *dialog* bit-vector **g**); 2. *replay* of the dialog (Algorithm 4, transforms y in-place into yx mod q); 3. uncomputation of the dialog (reverse of the first step). Note that [12] does not use the term "dialog" but we keep it from [79], since this is the same method. The

Subroutine	Calls	Operation	Subroutine scope	Out of scope
ComputeExponent	$(\lvert P\rvert+1)\lceil\frac{n_e}{w_1}\rceil$	Lookup$(w_1, \ell+\log(n_e))$	$n_e+\ell+\log(n_e)-w_1$	κ
		Q-Q Adder$(\ell+\log(n_e))$	n_e	
		$M_X(\ell+\log(n_e))$	$n_e+\ell+\log(n_e)$	
ReduceExponent	$\lvert P\rvert\log(n_e)$	Q-C Adder$(\ell+1)$	$\log(n_e)-1$	$n_e+\kappa$
		$CX(\ell)$	$\ell+\log(n_e)-1$	
		Q-Q Adder(ℓ)	$\log(n_e)$	
		$CX(\ell)$	$\ell+\log(n_e)-1$	
ComputeResidue	$\lvert P\rvert$	Lookup$(2w_3, \ell)$ (loading)	$\ell-2w_3$	$n_e+\log(n_e)+\kappa$
	$\lvert P\rvert\left(\lceil\frac{\ell}{w_3}\rceil-2\right)$	$P_{\lvert 0\rangle}(\ell)$	2ℓ	
	$\lvert P\rvert\left(\lceil\frac{\ell}{w_3}\rceil-2\right)\lceil\frac{\ell}{w_3}\rceil$	Lookup$(2w_3, \ell+1)$	$3\ell-2w_3+1$	
		Q-Q Adder$^\dagger(\ell+1)$	2ℓ	
		$M_X(\ell+1)$	$3\ell+1$	
		$CX(\ell)$	3ℓ	
		Q-Q Adder(ℓ)	$2\ell+1$	
		$CX(\ell)$	3ℓ	
		$M_X(1)$	3ℓ	
	$\lvert P\rvert\left(\lceil\frac{\ell}{w_3}\rceil-2\right)$	$M_X(\ell)$	2ℓ	
AccumulateResidue	$\lvert P\rvert\lceil\frac{\ell}{w_4}\rceil$	Lookup$(w_4, \kappa+1)$	$\ell-w_4+\kappa+1$	$n_e+\log(n_e)+\ell$
		Q-Q Adder$^\dagger(\kappa+1)$	ℓ	
		$M_X(\kappa+1)$	$\ell+\kappa+1$	
		$CX(\kappa)$	$\ell+\kappa$	
		Q-Q Adder(κ)	$\ell+1$	
		$CX(\kappa)$	$\ell+\kappa$	
	$\frac{\lvert P\rvert}{2}\lceil\frac{\ell}{w_4}\rceil$	Lookup(w_4, κ)	$\ell-w_4+\kappa$	
		Q-Q Comparison(κ)	ℓ	
		$M_X(\kappa)$	$\ell+\kappa$	
	$\lvert P\rvert\lceil\frac{\ell}{w_4}\rceil$	Phaseup(w_4)	$\ell-w_4+\kappa$	
ComputeResidue†	$2\lvert P\rvert\left(\lceil\frac{\ell}{w_3}\rceil-2\right)\lceil\frac{\ell}{w_3}\rceil$	Lookup$(2w_3, \ell+1)$	$3\ell-2w_3+1$	$n_e+\log(n_e)+\kappa$
		Q-Q Adder$^\dagger(\ell+1)$	2ℓ	
		$M_X(\ell+1)$	$3\ell+1$	
		$CX(\ell)$	3ℓ	
		Q-Q Adder(ℓ)	$2\ell+1$	
		$CX(\ell)$	3ℓ	
	$\lvert P\rvert\left(\lceil\frac{\ell}{w_3}\rceil-2\right)\lceil\frac{\ell}{w_3}\rceil$	Lookup$(2w_3, \ell)$	$3\ell-2w_3$	
		Q-Q Comparison(ℓ)	2ℓ	
		$M_X(\ell)$	3ℓ	
	$2\lvert P\rvert\left(\lceil\frac{\ell}{w_3}\rceil-2\right)\lceil\frac{\ell}{w_3}\rceil$	Phaseup$(2w_3)$	$3\ell-2w_3$	
	$\lvert P\rvert$	Phaseup$(2w_3)$ (unload)	ℓ	
ReduceExponent†	$\lvert P\rvert\log(n_e)$	Q-C Adder$(\ell+1)$	$\log(n_e)-1$	$n_e+\kappa$
		$CX(\ell)$	$\ell+\log(n_e)-1$	
		Q-Q Adder(ℓ)	$\log(n_e)$	
		$CX(\ell)$	$\ell+\log(n_e)-1$	
Last Phaseup	$\lceil\frac{n_e}{w_1}\rceil$	Phaseup(w_1)	n_e-w_1	κ

Table 15: Symbolic logical qubit count at each point of the approximate modular exponentiation circuit, and number of calls for each operation. "Out of scope" corresponds to the number of logical qubits not involved in the subroutine, while "subroutine scope" corresponds to the number of logical qubits involved elsewhere in the subroutine but not in the current operation. $CX(x)$ is a multi-target Cnot gate with x target logical qubits, $M_X(x)$ is the measurement in the X basis of x logical qubits and $P_{|0\rangle}(x)$ is the preparation of x logical qubits in the $|0\rangle$ state. In ComputeResidue, AccumulateResidue and ComputeResidue†, the $\log(n_e)$ term of the out-of-scope count is the quotient register.

Symbol	Definition	Role	secp256k1
q	prime defining the base field, $q = 2^u - f$	classical	$2^{256} - 4294968273$
f	pseudo-Mersenne defect $2^u - q$	classical	$2^{32} + 977$
λ_f	bit-size of f	classical	33
n	bit-size of q	classical	256
P, Q	base point and target point, $Q = [k]P$	classical	—
k	discrete logarithm of Q in base P	output	—
w	window length of the point addition	parameter	16
N_A	number of windowed point additions, $2\lceil n/w \rceil - 4$	classical	28
i	window index (address register)	quantum	—
c	flag ($i \neq 0$)	quantum	—
(x_1, y_1)	coordinates of the accumulator point	quantum	—
(x_2, y_2)	coordinates of the loaded point P_i	quantum, from a table	—
λ	slope $(y_1 - y_2)(x_1 - x_2)^{-1} \bmod q$	quantum	—
u, v	registers of the Euclidean algorithm	quantum	—
b_0, $b_0 \& b_1$	decision bits of one Euclidean iteration	quantum	—
g	garbage ("dialog") bit-vector	quantum	—
r, s	registers of the Bézout replay	quantum	—
c_{iter}	padding-iteration constant	parameter	2.4
c_{pad}	register-padding constant	parameter	2.3
N_{it}	gcd iterations, $3\lceil (1.413n + c_{\text{iter}}\sqrt{n})/3 \rceil$	parameter	402
G	garbage register, $5N_{\text{it}}/3 \simeq 2.355n$	classical	670
p	register padding, $\lceil c_{\text{pad}}\sqrt{n} \rceil$	parameter	37
B	shared u/v/garbage block, $G + 2p + 2\,\text{STEP}$	classical	748
A	ancilla budget of the hybrid adder, $n + 5 - (B - G)$	classical	183
ν_j	width of u, v at iteration j	classical	$n \to 10$
γ_j	garbage written after j iterations, $5\lceil j/3 \rceil$	classical	$0 \to 670$
τ	width of the truncated gcd comparison, $\text{TRUNCATE} + p$	parameter	77
μ_R	MSBs of the replay comparisons	parameter	40
φ_R	LSBs of the replay constant additions, $\mu_R + \lambda_f$	parameter	73
μ_S	MSBs of the square-subtract comparisons	parameter	50
φ_S	LSBs of the square-subtract constant additions, $\mu_S + \lambda_f$	parameter	83
N_Q	replay iterations handling $x + y = q$	parameter	50
a	residual $O(1)$ ancillae at the peak	classical	$\simeq 10$

Table 16: Parameters of the elliptic-curve discrete logarithm circuit, with the values used for secp256k1.

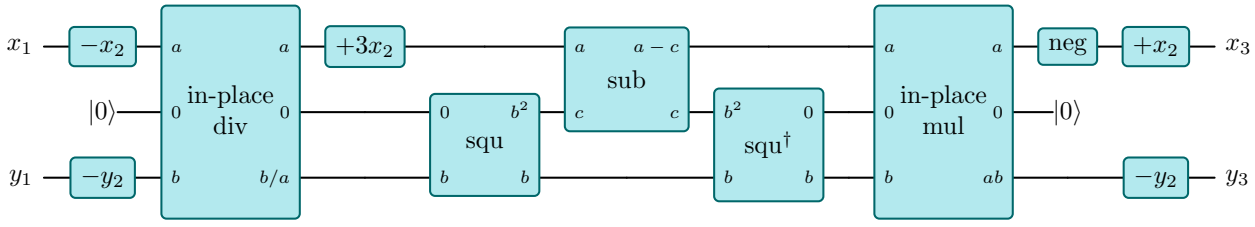


Figure 35: Arithmetic circuit for an elliptic-curve point addition (Algorithm 2). The small labels give the register contents before and after each block. Colors are as in Figure 26.

reverse of Steps 1 to 3 transforms y into $yx^{-1} \bmod q$.

The computation of the dialog follows a binary GCD algorithm (not extended), in which the input x is consumed and the dialog bit-vector $\mathbf{g}$ is produced. Each iteration records two Boolean values $(b_0, b_0 \& b_1)$ which indicate which controlled operations were performed.

As described in [12], an important optimization of the logical circuit is that the registers u, v are truncated throughout the iterations. Their most significant bits are freed and reused as storage for g. This optimizes the total qubit count and the gate count.

The (controlled) comparison used to compute $b_0 \& b_1$ is only performed on the τ most significant bits of u and v.

The replay step uses the dialog bits to reconstruct the Bézout coefficients from the extended GCD, asynchronously (Figure 37). In this algorithm, operations need to be performed modulo q.

In both algorithms, the storage of $(b_0, b_0 \& b_1)$ into $\mathbf{g}$ should be optimized, in order to minimize space (eventually the storage of $\mathbf{g}$ during the replay is the dominating space of the circuit). [12] does this with a *compression-decompression* circuit which packs three iterations into 5 bits. The gate count of this circuit is negligible throughout the algorithm.

B.2.4 Additive modular operations

While integer operations (subtraction and comparison) are immediately implemented, modular operations require another level of optimization. For these optimizations, we follow [12] again. Some of them are specific to the pseudo-Mersenne prime $q = 2^{256} - (2^{32} + 977)$ used in the secp256k1 standard curve. These optimizations add some arithmetic errors in the circuit, which are also bounded away by choosing appropriate amounts of *padding*. A padding of τ bits ensures that, assuming uniformly random inputs, an operation fails with probability $2^{-\tau}$.

i Modular Doubling. The circuit for modular doubling, in the case of $q = 2^{256} - (2^{32} + 977)$ and using 40 bits of padding, is represented in Figure 34. The value of $x \bmod q$ is first shifted by one bit, and the modular reduction is performed if the most significant bit is 1. It is approximated by simply adding $(2^{32} + 977)$ to the least significant bits (this number has 33 bits, and the window is expanded by 40 bits to allow for carry propagation). The most significant bit is erased by XORing the least significant bit (which is 1 if and only if a modular reduction occurred).

ii Controlled Modular Addition. The controlled modular adder for $q = 2^{256} - (2^{32} + 977)$ follows the same principle and can also use 40 bits of padding; it is represented in Figure 36. On this figure the bits of the inputs x and y have been reordered for clarity. A controlled adder on 257 bits is first performed, then the MSB of y controls the modular reduction (performed by adding $2^{32} + 977$ to the least significant bits of y). A comparison between x and y, which uses only their 40 MSBs, erases the MSB of y.

This circuit works only if the inputs x, y are taken uniformly at random. A slightly more costly variant of this circuit is used in [12] for the 50 first iterations of the replay step, in which the input may satisfy $x + y = q$ with high probability (this corresponds to "empty" dialog steps, needed since the dialog runs for a fixed number of iterations regardless of its input).

iii Constant addition. The additions by the constant $(2^{32} + 977)$ are performed using Gidney's low-space constant adder [91].

B.2.5 Logical cost breakdown

Tables 17 and 18 give, for each step of the windowed point addition, the number of calls over a whole run of Shor's algorithm and the logical qubit count, in the same format as Table 15 for factoring.

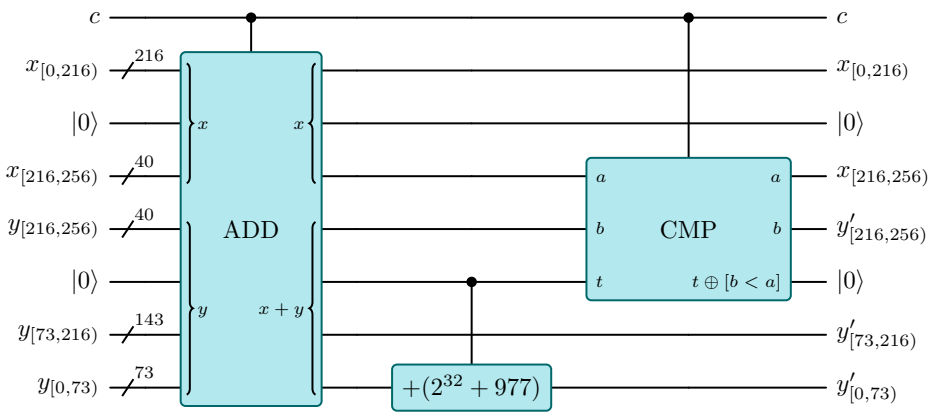


Figure 36: Optimized controlled modular addition for $q = 2^{256} - (2^{32} + 977)$ with 40 bits of padding. The two input ancillae $|0\rangle$ are where the MSBs of the ADD output are written (the first one remains 0, the second one becomes 1 if there is an overflow). The circuit computes $y' = y + c \cdot x \bmod q$.

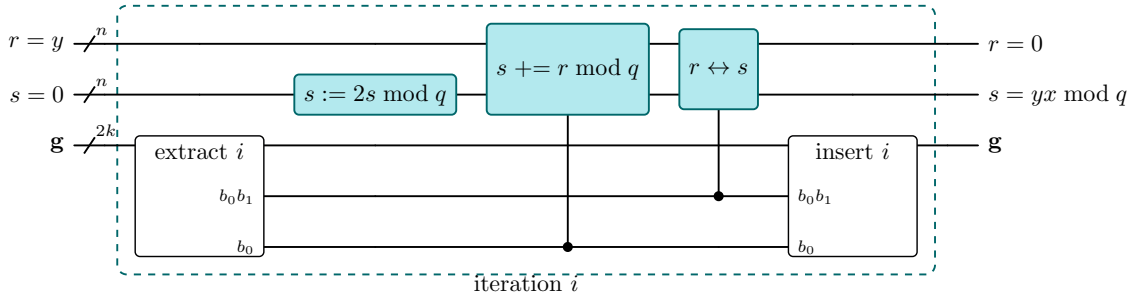


Figure 37: One step i of the dialog replay, pass 2 of Algorithm 4. Extract i moves the two bits recorded at iteration i, b_0 and $b_0 \wedge b_1$ (written $b_0 b_1$), out of the dialog g, and insert i puts them back, so the replay leaves g unchanged. The steps run from the last iteration to the first. Starting from $(r, s) = (y, 0)$, the replay ends at $(0,\ yx \bmod q)$. The dialog is erased afterwards by running pass 1 (Figure 33) backwards, which also restores (u, v).

Subroutine	Calls	Operation	Subroutine scope	Out of scope
$c \leftarrow (i \neq 0)$	$4N_A$	MC$X(w)$	$2n$	0
$-x_2,\ -y_2$	N_A	Lookup$(w, 2n)$	$2n$	0
	$2N_A$	Q-Q Adder$^\dagger(n{+}1)$	$2n + w - 2$	
	$2N_A$	Q-C Adder(φ)	$4n + w - \varphi$	
	$2N_A$	Q-Q Comparison(μ)	$4n + w - 2\mu$	
	N_A	$M_X(2n)$	$2n + w$	
	N_A	Phaseup(w)	$2n$	
inplace div inplace mul	$4N_A$	$P_{\|0\rangle}(B - 2n)$	$3n + 2$	w
	$4N_A N_{\text{it}}$	C$X(1)$	$B + n$	
	$4N_A N_{\text{it}}$	Q-Q Comparison(τ) (ctrl b_0)	$B + n - 2\tau$	
	$4N_A N_{\text{it}}$	CSWAP(ν_j) (ctrl $b_0 \& b_1$)	$B + n + 1 - 2\nu_j$	
	$4N_A N_{\text{it}}$	Q-Q Adder$^\dagger(\nu_j)$ (ctrl b_0)	$B + n + 1 - 2\nu_j$	
	$4N_A N_{\text{it}}$	Absorber	$B + n - 5$	
	$2N_A$	$P_{\|0\rangle}(n)$	$G + n + a$	
	$4N_A N_{\text{it}}$	Swapper	$G + 2n + a - 8$	
	$2N_A N_{\text{it}}$	Q-C Adder(φ_R) (ctrl)	$G + 2n + a - \varphi_R - 1$	
	$2N_A N_{\text{it}}$	Q-Q Adder$(n{+}1)$ (ctrl b_0)	$G + a - 3$	
	$2N_A N_{\text{it}}$	Q-C Adder(φ_R) (ctrl)	$G + 2n + a - \varphi_R - 1$	
	$2N_A N_{\text{it}}$	Q-Q Comparison(μ_R) (ctrl b_0)	$G + 2n + a - 2\mu_R - 2$	
	$2N_A N_{\text{it}}$	CSWAP(n) (ctrl $b_0 \& b_1$)	$G + a - 1$	
	$4N_A N_Q$	MC$X(\mu_R)$	$G + 2n + a - \mu_R - 1$	
	$2N_A N_Q$	C$X(n)$	$G + n + a - 1$	

Table 17: Symbolic logical qubit count at each point of the windowed elliptic curve point addition circuit (first half, continued in Table 18), and number of calls for each operation over a whole run of Shor's algorithm.

Subroutine	Calls	Operation	Subroutine scope	Out of scope
$+3x_2$	N_A	LOOKUP(w, n)	n	n
	N_A	Q-Q ADDER$(n{+}1)$	$w-2$	
	N_A	Q-C ADDER(φ)	$2n+w-\varphi$	
	N_A	Q-Q COMPARISON(μ)	$2n+w-2\mu$	
	N_A	$M_X(n)$	$n+w$	
	N_A	PHASEUP(w)	n	
square-subtract (ctrl c)	$2N_A(n{-}1)$	Q-C ADDER(φ_S) (ctrl)	$2n+1-\varphi_S$	w
	$2N_A(n{-}1)$	$CX(1)$	$2n$	
	$2N_An$	AND (ctrl c)	$2n-1$	
	N_An	Q-Q ADDER$(n{+}1)$ (ctrl)	1	
	N_An	Q-C ADDER(φ_S) (ctrl)	$2n+2-\varphi_S$	
	N_An	Q-Q COMPARISON(μ_S) (ctrl)	$2n+1-2\mu_S$	
neg (ctrl c)	N_A	$CX(n)$ (ctrl c)	0	$n+w$
	N_A	Q-C ADDER(φ) (ctrl c)	$n-\varphi$	
$+x_2,\ -y_2$	N_A	LOOKUP$(w, 2n)$	$2n$	0
	$2N_A$	Q-Q ADDER$(n{+}1)$	$2n+w-2$	
	$2N_A$	Q-C ADDER(φ)	$4n+w-\varphi$	
	$2N_A$	Q-Q COMPARISON(μ)	$4n+w-2\mu$	
	N_A	$M_X(2n)$	$2n+w$	
	N_A	PHASEUP(w)	$2n$	
Window recycling	N_A	$M_X(w)$	$2n$	0
	N_A	$P_{\|0\rangle}(w)$	$2n$	

Table 18: Symbolic logical qubit count at each point of the windowed elliptic curve point addition circuit (second half, continuing Table 17)

.

C Resource estimation for RSA-2048

C.1 Architecture

C.1.1 Layout, connectivity and noise

In the repetition-cat architecture, data qubits lie on a square lattice and ancillae at the centers of the corresponding plaquettes (Figures 43 and 46). Outside the factories, the physical operations are preparation in $|0\rangle$ or $|+\rangle$, CNOT gates between neighboring qubits, and X or Z measurements. Each ancilla is coupled to the four data qubits at the corners of its plaquette, such that every qubit has degree four connectivity. Transversal CNOT gates between repetition codes laid out in adjacent columns require CNOT gates between horizontally neighboring data qubits, which are not coupled. We emulate them with the degree-four connectivity. Each CNOT gate between two horizontally neighboring data qubits is mediated by an ancilla adjacent to both. The ancilla is prepared in $|0\rangle$, receives a CNOT gate from the control, applies a CNOT gate onto the target, and is measured in the X basis. The outcome is absorbed as a Z correction on the control in the Pauli frame. This sequence has the structure, and hence the duration, of a stabilizer round, and each single fault still causes at most one phase flip per repetition code. We assume that all physical qubits are identical, such that any qubit of this lattice can serve as a data qubit or an ancilla.

The LDPC-cat architecture [16] is a flip chip of two facing layers (Figure 39). The *repetition layer* has the lattice and connectivity above. The *LDPC layer* holds the data qubits of an LDPC-cat code (Section C.1.3) on the same square lattice, and one ancilla per check, coupled to the at most four data qubits of its check, which fit within a 3×3 grid of neighboring data qubits. Each data qubit of the LDPC layer is coupled to at most four ancillae, and to the facing qubit of the repetition layer. The two layers do not correspond to two monolithic chips. To scale to the required number of qubits, the device would be split into chiplets, coupled either directly across separate dies [92] or through cryogenic microwave links [93, 94]. Such links are noisier than on-chip couplings and could therefore add an overhead, which recent works [95, 96] suggest can remain small given a sufficient rate of shared Bell pairs and a remote link infidelity of the order of a percent. The partitioning into chiplets and the cost of inter-chip links are beyond the scope of this work.

We denote the durations of a transversal CNOT gate, local initialization, destructive measurement, and one stabilizer round by T_{CNOT}, T_{init}, T_{meas}, and T_{round}, respectively. One stabilizer round defines an error-correction cycle, $T_{\text{round}} = 1$ cycle $= 1\,\mu\text{s}$. A logical measurement outcome is available for feedforward after the decoder reaction time T_{react}. We take $T_{\text{CNOT}} = T_{\text{init}} = T_{\text{meas}} = 1$ cycle and $T_{\text{react}} = 10$ cycles. These durations hold in both layers, including for transversal inter-layer CNOT gates. Durations are rounded up to whole cycles. Moreover, we assume a noise model neglecting bit flips. At the main point of Section C.7.2 (code distance 15), one RSA-2048 shot involves about 1.2×10^{15} physical qubit-cycles (49 649 qubits $\times$ 2.49×10^{10} error-correction cycles), or a bit-flip probability well below 10^{-15} per qubit and cycle. We treat this as a working hypothesis (Section 3), which we quantify in Section C.9.3 and mitigate in Section C.9.4. Phase flips follow the generic circuit-level noise model of [16, Supplementary Table 1(b)], with a single parameter p_Z. Each initialization in $|+\rangle$ is followed by a Z error with probability p_Z, and each X-basis measurement outcome is flipped with probability p_Z. Each CNOT gate is followed by Z_c, Z_t or Z_cZ_t with probability $p_Z/3$ each, and every qubit idle during an operation undergoes a Z error with probability p_Z. Phase flips have no effect on preparations in $|0\rangle$ or on Z-basis measurements. In the following, we consider two phase-flip probabilities, $p_Z = 10^{-3}$ and $p_Z = 5 \times 10^{-4}$, and unless stated otherwise, use the latter.

C.1.2 Zones

The chip is divided into zones whose number and arrangement depend on the subroutine and on the architecture: *compute*, *buffer*, *routing* and *factory* zones for the ADDER (Figure 43), and further *exponent* and *memory* zones in the complete layouts (Figures 50 and 53). A zone is a region of the repetition layer, extended on the LDPC-cat architecture by the facing region of the LDPC layer (Section C.1.3). The states $|\mathrm{C}_X\mathrm{C}_X X\rangle$ are produced by a two-level factory with a shared first-level production stage [47] and $|i\rangle$ states from color code factories [47, 97, 98] that store their produced states alternately into two output repetition codes. Their parameters are given in Table 19.

C.1.3 Error correcting codes

In the repetition-cat architecture, logical qubits are stored in repetition codes of distance d, each made of d data qubits and $d-1$ ancillae ($2d-1$ physical qubits). The stabilizers of each code are X_iX_{i+1}, its codewords are $|\pm\rangle_L \coloneqq |\pm\rangle^{\otimes d}$, and its logical operators are $X_L = X_1$, which has weight one, and $Z_L = Z^{\otimes d}$, which has weight d. The logical preparation $|+\rangle_L$ of a repetition code is performed at the physical level, by initializing its data qubits in $|+\rangle$ in T_{init}. The preparation $|0\rangle_L$ also initializes the data qubits, in $|0\rangle$, in T_{init}. Its first stabilizer outcomes are random and only define a Pauli frame. Establishing it takes about d rounds, which must precede any use of an X-measurement outcome of the code, or of a code coupled to it by transversal CNOT gates. When a following operation provides these rounds, the preparation adds no time to them. Otherwise, it lasts d error-

correction cycles. On the repetition-cat architecture, in the color-changed convention (Section C.3), every $|0\rangle_L$ preparation is followed by a lattice surgery (with one harmless exception in the auto-AND preparation, Section C.5.4), so none lasts d.

In the LDPC-cat architecture, the LDPC layer stores logical qubits in an open-boundary LDPC-cat code of distance d_{LDPC} (Table 20). Its upper rows extend laterally into a left and a right *staircase*, which add Δ_{LDPC} data qubits so that the outermost logical qubits keep the full distance. The region of the repetition layer facing it, the *LDPC layer access zone*, provides access to both logical operators of every logical qubit of the LDPC layer. The operator X_L is supported on a single *tip* qubit, which lies in the bottom rows of the code. The facing qubit of the repetition layer serves as a *leaf* ancilla and measures the X parity of the tip and of a *proxy* data qubit, through which X_L is then accessible. The operator Z_L, supported on at least d_{LDPC} qubits from the tip upward, is accessed through transversal inter-layer CNOT gates onto a repetition code covering its support. In this architecture, when necessary, we distinguish the distance d_{rep} of the repetition codes of the repetition layer from the distance d_{LDPC} of the LDPC-cat code. As is common in the literature [99], we count the duration of each lattice surgery step as

$$T_{\mathrm{surg}} = d\,T_{\mathrm{round}}. \tag{77}$$

A lattice surgery confined to the repetition layer lasts T_{surg} of Equation (77) with $d = d_{\mathrm{rep}}$, whereas a lattice surgery involving the LDPC layer conservatively lasts

$$T_{\mathrm{surg}}^{(\mathrm{LDPC})} = d^\star\,T_{\mathrm{round}}, \qquad d^\star = \max(d_{\mathrm{rep}}, d_{\mathrm{LDPC}}), \tag{78}$$

such that its timelike distance is at least the spatial distance of every code that takes part in it. In particular, an access to X_L through its leaf ancilla, or to Z_L through a covering repetition code that receives the transversal inter-layer CNOT gates during its first stabilizer round, lasts $T_{\mathrm{surg}}^{(\mathrm{LDPC})}$.

C.2 Lattice-surgery primitives

C.2.1 Parity-CNOT gate on cat qubits

Given $N \geq 1$ and $x_1, \ldots, x_N, y \in \{0,1\}$, the parity-CNOT gate is defined by the following operation:

$$\begin{aligned} &\mathrm{PC}X(N)\ |x_1 \ldots x_N\rangle\,|y\rangle \\ &\quad = |x_1 \ldots x_N\rangle\ |x_1 \oplus \cdots \oplus x_N \oplus y\rangle\,. \end{aligned} \tag{79}$$

The repetition-cat implementation, shown in Figure 38(a), is adapted from the long-range CNOT scheme of [4]. CNOT gates applied from $x_1, \ldots, x_N$ onto an auxiliary qubit prepared in $|0\rangle_L$ collect the parity $x_1 \oplus \cdots \oplus x_N$, and an XX parity measurement between the auxiliary qubit and the target, followed by a destructive Z measurement of the auxiliary qubit, completes the parity-CNOT gate, up to Pauli corrections. This operation is performed in approximately one lattice surgery step, or T_{surg} cycles (Equation (77)), independently of N.

The gate layout is asymmetric. It accesses each control through the logical Z_L representative, supported on all the d data qubits, and the target through the logical X_L representative, supported on any single data qubit. The auxiliary qubit is therefore encoded in a tree repetition code with one branch along each control, which reaches the target at a single qubit (Figure 38(a)). Transversal CNOT gates from the d data qubits of a repetition code onto any d neighboring data qubits of a tree repetition code implement a logical CNOT gate in the same direction. As such, the N CNOT gates act on disjoint qubits and are applied in a single step. Additional controls therefore cost about d routing data qubits each, but no additional time.

The scheme generalizes to the LDPC-cat architecture [47], where the auxiliary qubit is encoded in a *snake* repetition code covering, in the repetition layer, the support of the product of the LDPC logical representatives $Z_{L,x_1}, \ldots, Z_{L,x_N}$ of the controls. The CNOT gates are then transversal inter-layer gates, applied in a single step, and the snake is merged with the target through the XX measurement, then measured out in the Z basis.

C.2.2 Multi-target CNOT gate on cat qubits

Similarly, given $x, y_1, \ldots, y_N \in \{0,1\}$, the multi-target CNOT gate implements the following operation:

$$\begin{aligned} &\mathrm{C}X(N)\ |x\rangle\,|y_1 \ldots y_N\rangle \\ &\quad = |x\rangle\ |x \oplus y_1, \ldots, x \oplus y_N\rangle\,. \end{aligned} \tag{80}$$

By the asymmetry of the repetition code, the multi-target CNOT gate, presented in [4], is implemented differently from the parity-CNOT gate. A CNOT gate is applied from x onto an auxiliary qubit prepared in $|0\rangle_L$, and an $XX \cdots X$ parity measurement between the auxiliary qubit and the targets, followed by a destructive Z measurement of the auxiliary qubit, completes the multi-target CNOT gate, up to Pauli corrections.

The gate accesses the control once, through the logical Z_L representative supported on all the associated d data qubits, and each target only through its logical X_L representative supported on any single data qubit. On the repetition-cat architecture, a single lattice surgery merge measures the parity [4] (Figure 38(b)). Additional targets therefore cost routing, which scales independently of d, but does not require additional time.

The scheme extends to the LDPC-cat architecture, where the auxiliary qubit is encoded in a snake repetition code covering, in the repetition layer, the support of the LDPC logical representative $Z_{L,c}$ of the control c. The CNOT gate is then a transversal inter-layer

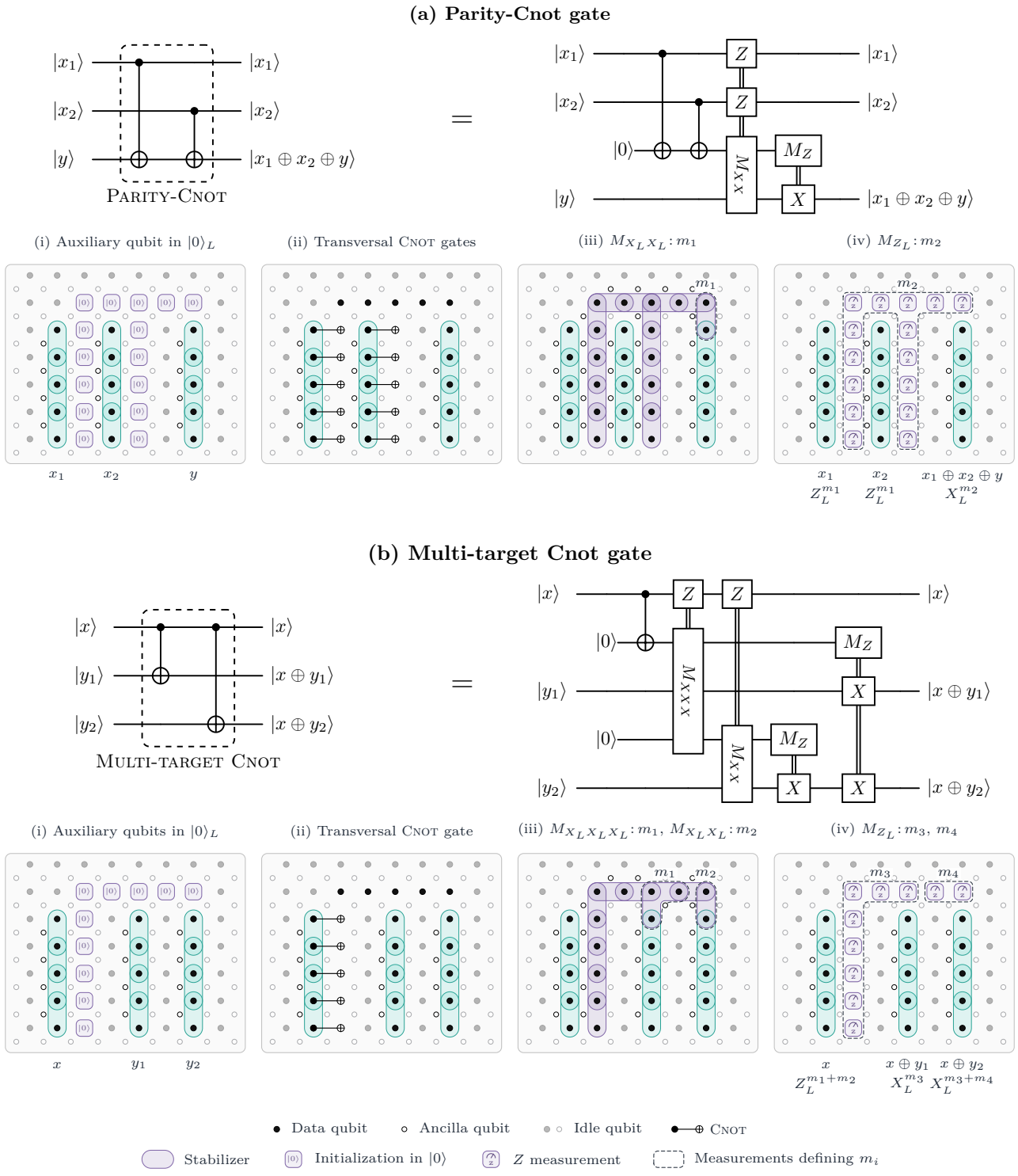


Figure 38: Implementation of the parity-Cnot gate (a) and of the multi-target Cnot gate (b) on the repetition-cat architecture. Top of each panel: logical gate and equivalent implementation circuit. Bottom: repetition-cat layout. (i) The auxiliary qubits are prepared in $|0\rangle_L$ (Section C.1.3). In practice, each data qubit (in purple online) is initialized in $|0\rangle$. Since their stabilizers commute with the transversal Cnot gates of (ii), these are measured for the first time in (iii), in order to save time. For simplicity, the syndromes of the auxiliary qubits are assumed to be trivial. (ii) Transversal Cnot gates from each control to the neighboring branch of the auxiliary qubit. (iii) A lattice surgery merge [100] measures $X_L X_L$ (m_1) on the auxiliary qubit and the target (a), or $X_L X_L X_L$ (m_1) and $X_L X_L$ (m_2) on the auxiliary qubits and the targets (b). The dashed contours mark the checks that give these outcomes. In (b), the last two routing qubits form the second auxiliary qubit. (iv) The auxiliary qubits are measured in the Z basis, with outcomes m_2 (a), or m_3 and m_4 (b), and Pauli corrections are applied in the Pauli frame.

gate, and the parity measurement reaches the targets through a *fork* code (Figures 39 and 40). If the fork operation can be performed adjacent to the snake, the gate requires approximately a single lattice surgery step (Figure 39). Otherwise, the snake is reduced, which requires an additional lattice surgery step and frees space for the fork measurement (Figure 40).

C.2.3 Active injection

When a Toffoli gate is performed by injecting $|\mathrm{CC}X\rangle$ or another CCZ-type state (i.e. any state obtained from $|\mathrm{CC}Z\rangle$ by applying a Hadamard gate to a subset of its three qubits), each injection CNOT gate can be replaced by a parity-CNOT gate or a multi-target CNOT gate. Measuring multi-qubit parities is natural in lattice surgery [99, 101]. We denote this procedure, adapted from [11], by *active injection*. In the following, C_X denotes a control in the X basis, so that $|\mathrm{C}_X\mathrm{C}_X Z\rangle \coloneqq H^{\otimes 3}|\mathrm{CC}X\rangle$ and $|\mathrm{C}_X\mathrm{C}_X X\rangle \coloneqq H^{\otimes 3}|\mathrm{CC}Z\rangle$ are the color-changed counterparts of $|\mathrm{CC}X\rangle$ and $|\mathrm{CC}Z\rangle$. Let I, J, K be sets of logical qubits with $I, J \neq \emptyset$, and let x_I denote the XOR of the qubits in I. The operations

$$\begin{aligned} U_{\mathrm{CC}X}\,|x\rangle\,|c\rangle &= |x\rangle\,|c \oplus x_I \cdot x_J \oplus x_K\rangle\,, \\ U_{\mathrm{CC}Z}\,|x\rangle &= (-1)^{x_I \cdot x_J \cdot x_K}\,|x\rangle \quad (K \neq \emptyset), \end{aligned} \tag{81}$$

where the target c is not among the inputs, can be implemented in the time of a single Toffoli (respectively CCZ) injection by lattice surgery. The terms x_I and x_J come from replacing the injection CNOT gates onto the control legs of the magic state by parity-CNOT gates (Figure 41). The term x_K comes from adding the qubits of K as extra controls of the CNOT gate from the target leg onto c (for $U_{\mathrm{CC}Z}$, of the parity-CNOT gate onto the third leg). The sets I, J and K may overlap. For example, the sum bit of Equation (86) is $U_{\mathrm{CC}X}$ with $c = 0$, $I = \{a_k, s_k\}$, $J = \{b_k, s_k\}$ and $K = \{a_k, b_k, s_k, a_{k+1}, b_{k+1}\}$.

On the target leg, the multi-target CNOT gate simplifies, since the leg is measured in the X basis right after. Indeed, a multi-target CNOT gate from the leg onto $c_1, \ldots, c_N$ followed by its X measurement amounts to the single X-parity measurement of the leg and $c_1, \ldots, c_N$, followed by a Z measurement of the leg, up to Pauli corrections. Figure 41 combines a parity-CNOT gate on each control leg with a multi-target CNOT gate on the target leg, implemented as this parity measurement. The resulting $|\mathrm{CC}X\rangle$ injection computes $c_j \oplus (a_1 \oplus a_2)(b_1 \oplus b_2)$ for $j = 1, 2$ in one lattice surgery step, followed by the associated Clifford corrections.

C.3 Color-changed implementation

Throughout this work, we implement the color-changed (Hadamarized) version of the computation, in which the roles of the X and Z bases are exchanged in every circuit relative to the standard presentation. This convention allows a more efficient implementation of the ADDER and LOOKUP operations, which dominate the cost of the algorithm (see Section C.5 and Ref. [47]). The physical code is unchanged, with a logical operator Z_L of weight d and a logical operator X_L of weight one. Exchanging the bases reverses the direction of every CNOT gate, exchanges parity-CNOT gates with multi-target CNOT gates, and swaps the preparation and measurement bases, such that $|+\rangle_L$ preparation and X-basis measurement in the standard circuits become $|0\rangle_L$ preparation and Z-basis measurement. Since all circuits are exchanged consistently, the logical operation implemented is unchanged. Unless stated otherwise, all circuits and figures of Sections C.5 to C.8 are drawn in the color-changed convention.

C.4 Magic-state factories and cadence

The non-Clifford operations of the computation, namely the ADDER, the LOOKUP and the PHASEUP, consume two types of resource states, $j \in \{|\mathrm{C}_X\mathrm{C}_X X\rangle, |i\rangle\}$, produced by dedicated factories, presented in [47]. In the following, these operations are decomposed into steps. Each step consumes at most one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ state and two $|i\rangle$ states, all loaded simultaneously for consumption in a buffer during a lattice surgery of duration T_{surg}. Unless stated otherwise, steps consuming factory outputs are applied at least a *cadence* T_{cad} apart.

The $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory is composed of two second-level factories of distance 19 sharing three pairs of first-level factories. The $|i\rangle$ factory is a distance-7 color code with two output repetition codes of distance 19. To fit the factory on the lattice of Section C.1.1, three of the six stabilizers along each output repetition code are measured by two ancillae sharing a Bell pair, which adds three qubits per output repetition code to the qubit count presented in [47, Table II(a)]. These qubits are included in Table 19, and their small effect on the simulated output error is neglected. We denote by l^j_{fac} and S^j_{fac} the length, in data qubit columns, and the footprint, in physical qubits, of one factory of type j, by T^j_{prod} its mean production time excluding loading, and by n^j_{fac} the number of such factories. These parameters are summarized in Table 19. Each factory produces a state sequentially, in a mean time T^j_{prod}, and alternates between its two output channels. We assume each channel is occupied from the start of a production to the end of its loading. Used cyclically and assigning each state of an $|i\rangle$ pair to a dedicated factory, a sufficient condition to sustain the consumption rate is

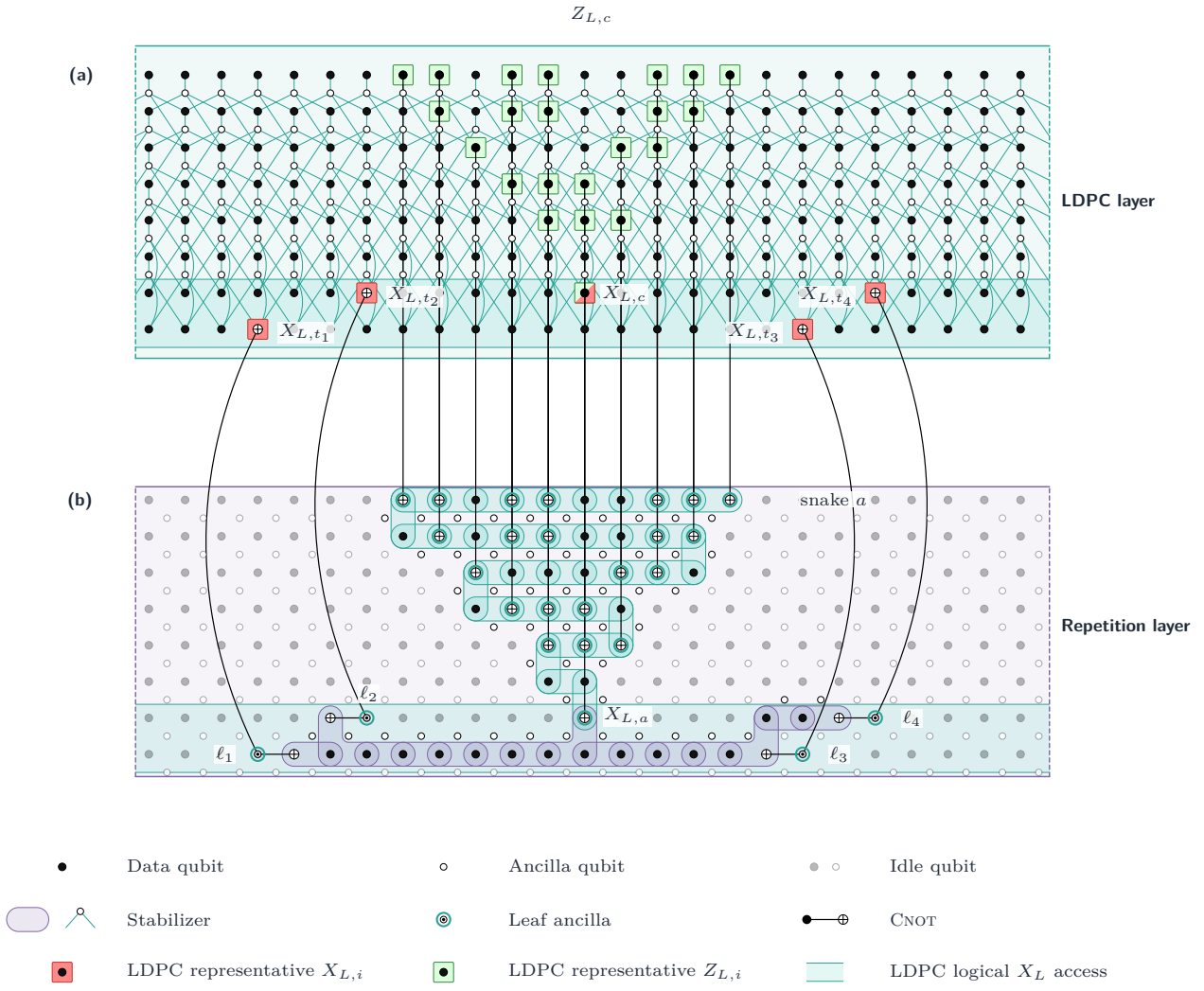


Figure 39: Multi-target CNOT gate on the LDPC-cat architecture, from a control c onto targets $t_1, \ldots, t_4$, in one lattice surgery step. (a) LDPC layer, with a logical representative $Z_{L,c}$ of the control and the weight-one logical representatives $X_{L,c}$ and X_{L,t_i}. (b) Repetition layer, with the auxiliary qubit a, encoded in a snake repetition code covering the support of $Z_{L,c}$, and a fork code with one auxiliary qubit per target. The fork joins the snake at $X_{L,a}$ and reaches each target through a leaf ancilla ℓ_i. The snake repetition code is prepared in $|0\rangle_L$ and the fork qubits in $|0\rangle$. The snake then receives the transversal inter-layer CNOT gates from the support of $Z_{L,c}$ in the second cycle. Finally, the lattice surgery merge measures $X_{L,a}X_{L,t_1}\cdots X_{L,t_4}$, after which the snake and the fork are measured in the Z basis, and Pauli corrections are applied in the Pauli frame.

Factory	Length l^j_{fac} (columns)	Footprint S^j_{fac} (qubits)	Production T^j_{prod} (cycles)	Output error $p_{Z,\text{out}}$ $p_Z = 10^{-3}$	Output error $p_{Z,\text{out}}$ $p_Z = 5\times10^{-4}$
$\|\mathrm{C}_X\mathrm{C}_X X\rangle$	29	1 566	21	1.48×10^{-11}	2.02×10^{-13}
$\|i\rangle$	10.5	153	7	3.1×10^{-12}	9.43×10^{-13}

Table 19: Factory parameters for $p_Z = 10^{-3}$ and $p_Z = 5\times10^{-4}$.

then given by

$$n^{|\mathrm{C}_X\mathrm{C}_X X\rangle}_{\text{fac}} = \left\lceil \frac{T^{|\mathrm{C}_X\mathrm{C}_X X\rangle}_{\text{prod}}}{T_{\text{cad}}} \right\rceil, \qquad n^{|i\rangle}_{\text{fac}} = 2\left\lceil \frac{T^{|i\rangle}_{\text{prod}}}{T_{\text{cad}}} \right\rceil, \tag{82}$$

where the cadence time is assumed at least as large as the loading time. In this work, we choose to minimize the factory footprint, with the cadence choice

$$T_{\text{cad}} = \max\left(T_{\text{surg}}, T^{|\mathrm{C}_X\mathrm{C}_X X\rangle}_{\text{prod}}, T^{|i\rangle}_{\text{prod}}\right). \tag{83}$$

Using the factories of Table 19, this cadence corresponds to $T_{\text{cad}} = \max(d, 21)$. Using one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory and two $|i\rangle$ factories is then sufficient at every distance. The $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory therefore sets the

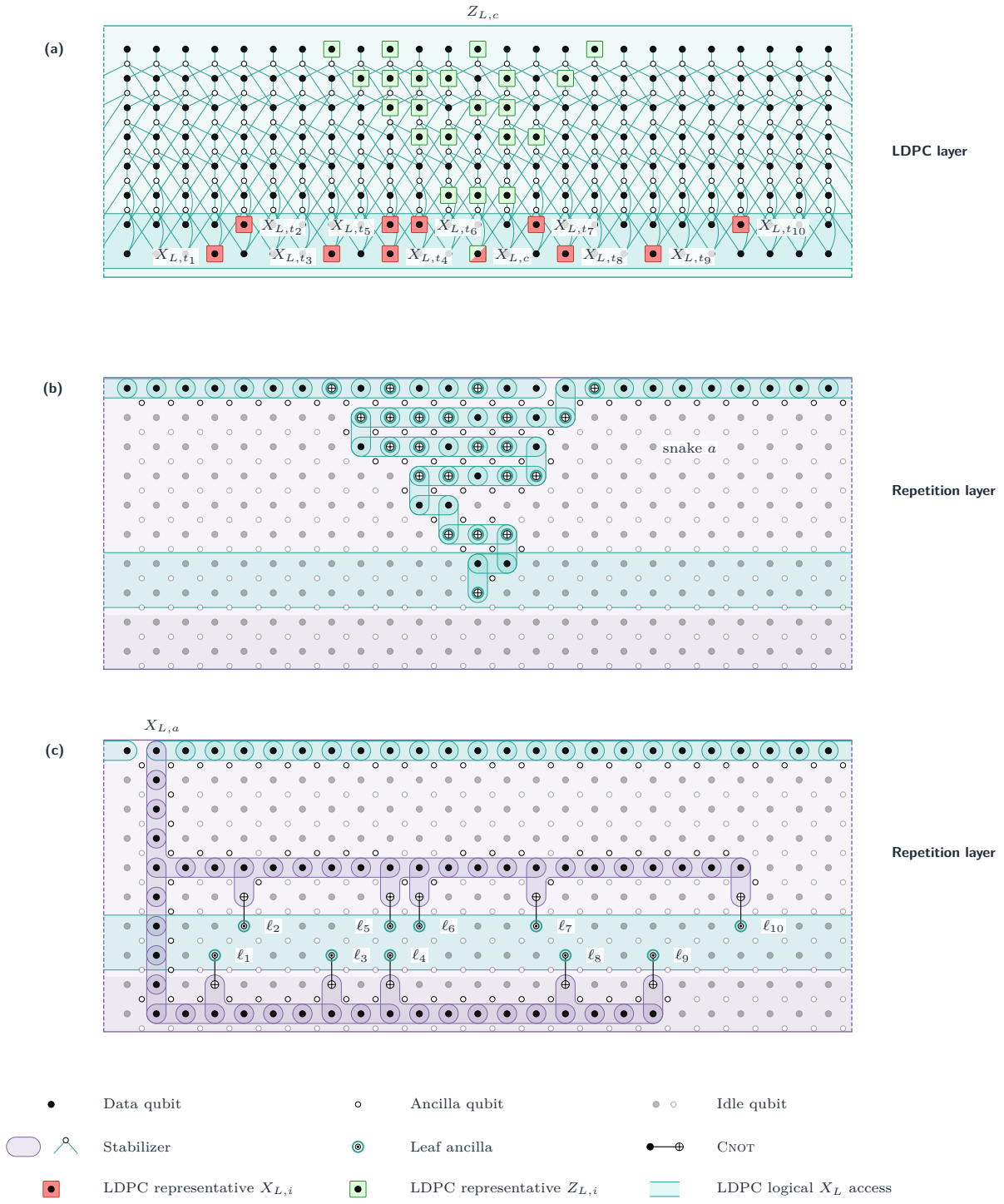


Figure 40: Multi-target CNOT gate on the LDPC-cat architecture, from a control c onto targets $t_1, \ldots, t_{10}$, in approximately two lattice surgery steps and two extra rows in the repetition layer (gray). (a) LDPC layer, with logical representatives $Z_{L,c}$, $X_{L,c}$ and X_{L,t_i}. (b) During the first lattice surgery step, the auxiliary qubit a, a snake repetition code prepared in $|0\rangle_L$, covers the support of $Z_{L,c}$ and the top row of the repetition layer, the tail, and receives the transversal inter-layer CNOT gates from the support of $Z_{L,c}$ in the second cycle. (c) During the second lattice surgery step, after Z measurements reduce the snake to its tail, the fork, with qubits prepared in $|0\rangle$, joins the tail at $X_{L,a}$ and measures $X_{L,a}X_{L,t_1}\cdots X_{L,t_{10}}$ through one leaf ancilla ℓ_i per target. The tail and the fork are then measured in the Z basis, and Pauli corrections are applied in the Pauli frame.

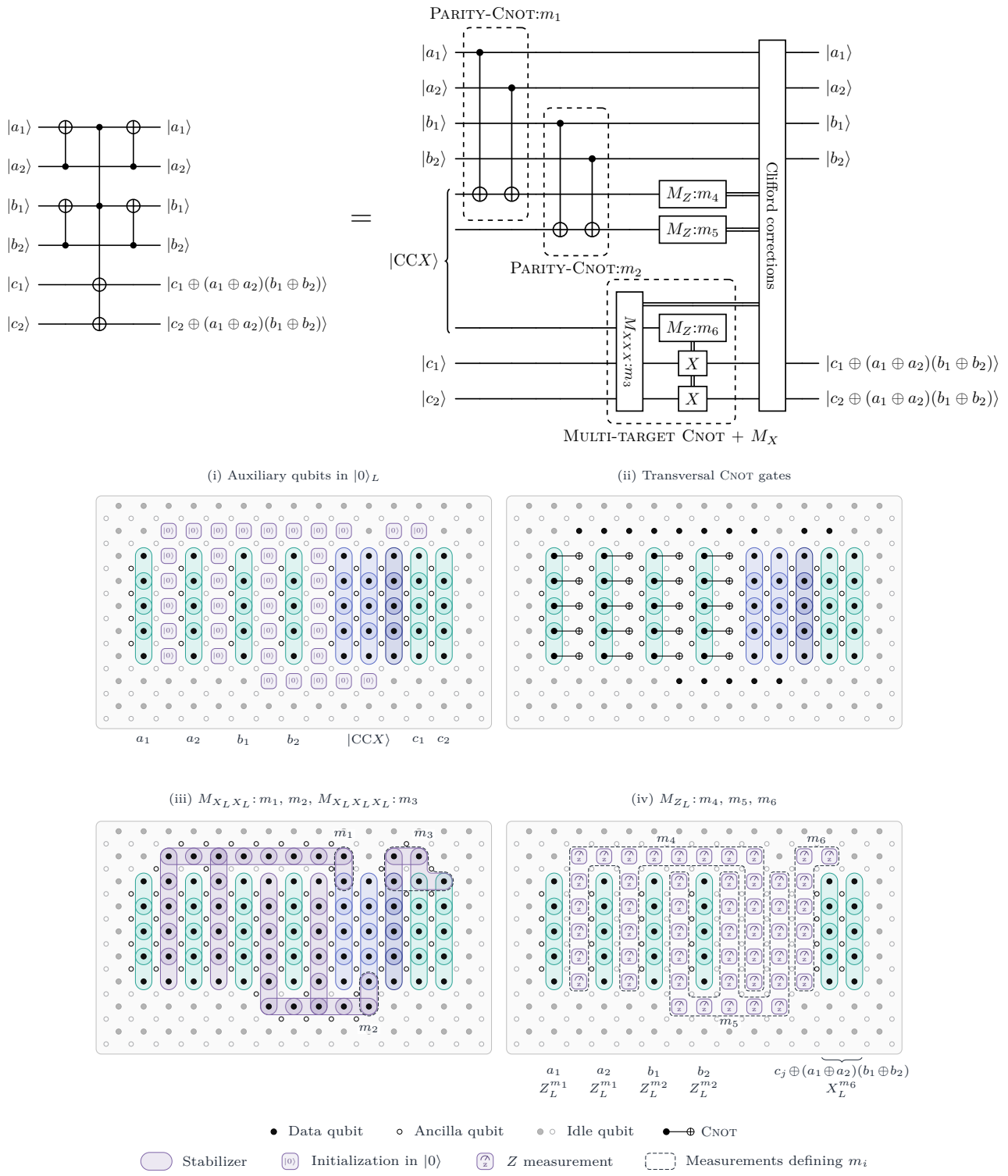


Figure 41: Active injection of $c_j \mapsto c_j \oplus (a_1 \oplus a_2)(b_1 \oplus b_2)$ for $j = 1, 2$, with parity-Cnot gates on the control qubits of a $|\mathrm{CC}X\rangle$ state and a multi-target Cnot gate on its target qubit. Top: logical gate and equivalent injection circuit involving the consumption of a resource state. The multi-target Cnot gate followed by the X_L measurement of the control (dashed) is implemented as the $X_LX_LX_L$ parity measurement and the destructive Z measurement of the target qubit, up to Pauli corrections. Bottom: repetition-cat layout, with the control qubits of $|\mathrm{CC}X\rangle$ in blue and the corresponding target qubit in dark blue. (i) The auxiliary qubits are prepared in $|0\rangle_L$, as in Figure 38. (ii) Transversal Cnot gates onto the routing trees of the parity-Cnot gates. (iii) A merge measures X_LX_L between each tree and the control qubit of $|\mathrm{CC}X\rangle$ (m_1, m_2, parity-Cnot gates) and $X_LX_LX_L$ on the target qubit of $|\mathrm{CC}X\rangle$, c_1 and c_2 (m_3, multi-target Cnot gate). (iv) Z measurements of the auxiliary qubits, whose outcomes enter those of the $|\mathrm{CC}X\rangle$ qubits they are merged with (m_4, m_5, m_6). Pauli corrections are applied in the Pauli frame.

cadence for small distances, while the lattice surgery sets it for large distances.

A step is *cadence-limited* when it is required to be applied at least T_{cad} cycles after the previous step. These are the steps of the ADDER (Section C.5.6), the AND gates of a LOOKUP not performed simultaneously with a parity-CNOT gate, and the AND gates of a PHASEUP (Section C.6). The steps that are *surgery-limited* last T_{surg}, namely the parity-CNOT gates (Section C.2.1), the multi-target CNOT gates (Section C.2.2), the AND gates of a LOOKUP performed simultaneously with a parity-CNOT gate, and the multi-target $\mathrm{C}_X X$ gates of a PHASEUP (Section C.6). Note that the multi-target $\mathrm{C}_X X$ gates consume only $|i\rangle$ states and can be applied sequentially every T_{surg} with the same two $|i\rangle$ factories, since $T_{\text{prod}}^{|i\rangle} \leq T_{\text{surg}}$ for the factories of Table 19.

C.5 ADDER

In this section, we present an adaptation of the adder introduced in [11] to cat-qubit architectures. Following the convention presented in Section C.3, register states are written in the X basis, such that we use the shorthand notation $|x\rangle_X \coloneqq H|x\rangle$. Unless explicitly stated otherwise, all states discussed below are logical states. Given two integers in binary representation $a, b \in \{0,1\}^n$, the circuit for the addition implements the following operation:

$$\text{ADDER}\,|a\rangle_X\,|b\rangle_X = |a\rangle_X\,|a+b \mod 2^n\rangle_X\,. \quad (84)$$

Using a carry $c \in \{0,1\}^n$, this computation can be decomposed into elementary operations by using the relations:

$$\begin{aligned} s_k &= a_k \oplus b_k \oplus c_k \\ c_{k+1} &= a_k b_k \oplus b_k c_k \oplus a_k c_k, \end{aligned} \quad (85)$$

with s the outcome of the ADDER, a_k, b_k, c_k and s_k the k-th bit of a, b, c and s, respectively. Alternatively, for an out-of-place addition, the output can be reused directly, without a carry, in order to reduce the number of non-linear operations as follows:

$$s_{k+1} = ((a_k \oplus s_k)\cdot(b_k \oplus s_k)) \oplus (a_k \oplus b_k \oplus s_k \oplus a_{k+1} \oplus b_{k+1}). \quad (86)$$

In practice, this implies that the k-th step of the addition can be implemented using a single non-Clifford AND gate on linear combinations of inputs, which the active injection of Section C.2.3 absorbs into the lattice surgery of the injection itself. The bit b_{k+1} is uncomputed by measurement, and the non-linear term of the resulting phase correction, given by the same identity, is a $\mathrm{C}_X X$ gate between $a_k \oplus s_k$ and $b_k \oplus s_k$. The remaining linear terms then correspond to a Pauli frame update.

C.5.1 Logical circuit

Based on the identity (86), the logical circuit design is adapted from the construction presented in Appendix A.2 of [11]. An AND gate injected from a $|\mathrm{CC}X\rangle$ state requires two outcome dependent CNOT corrections. If these corrections are applied after the injection, further computations must wait for the reaction time of the decoder, since they depend on its measurement outcomes. At the cost of additional qubits, it is possible to include all (or part) of these corrections through adaptive measurements on an offline *(semi-)auto* magic state [102]. Gidney [11] optimizes the adder circuit for surface code compilation by using *semi-auto*-AND states. These states allow AND gate injections requiring classically controlled phase gates on the control qubits. Remarkably, in their adder circuit, those phase gate byproducts can be later merged, with no additional cost, into the corrections required for the uncomputation of one of the input registers. Preparation of this semi-auto-AND state requires three $|i\rangle$ states, in addition to a $|\mathrm{CC}Z\rangle$ state and a Hadamard gate. This implementation is favorable for surface codes, since both the $|i\rangle$ state preparation [103] and the Hadamard gate are spatially and temporally efficient.

The cost balance is different for cat-qubit architectures. Indeed, $|i\rangle$ states are supplied by dedicated color code factories and a Hadamard gate consumes two $|i\rangle$ states, with a third reused as a catalyst [47, App. C 2]. We implement the AND gate with an auto-AND magic state prepared from a $|\mathrm{C}_X\mathrm{C}_X X\rangle$ state, and the subsequent phase corrections with *delayed*-$\mathrm{C}_X X$ states $|\mathrm{DC}_X X\rangle$. Each preparation consumes two $|i\rangle$ states and reuses a catalyst, reducing the factory throughput required between consecutive ADDER steps.

The circuit stores a and b in separate registers and writes the result into a third s register during injection, following the identity of Equation (86). We denote the width of each of the three registers by n and index the bits by $k = 1, \ldots, n$ from least to most significant. Each logical qubit is denoted by the bit it stores.

The ADDER performs n injections followed by n corrections in reverse order, $I_1, \ldots, I_n, C_n, \ldots, C_1$. At the k-th step, the injection I_k computes s_k and the correction C_k performs a destructive Z measurement on b_k and applies the corresponding phase correction. After the corrections, a and s are the remaining quantum registers. In the context of Appendix B, the register a corresponds to the output of a LOOKUP, the register b to the previous value of an accumulator, and the register s to the corresponding update of the latter. As such, uncomputing b leads to an in-place update, while a is cleared afterwards by the measurement-based uncomputation of the LOOKUP, whose phases are resolved by a PHASEUP (Section C.6).

Figure 42 provides the logical circuit implementing Equation (86) at $k > 1$. The X-parity measurements follow the active injection procedure of Section C.2.3. Section C.5.5 details the implementation of these measurements in a single simultaneous lattice surgery step [46]. The recorded measurement outcomes determine the adaptive X or Z measurement bases of the auto-And and delayed-$\mathrm{C}_X X$ states. Tracking the measurement outcomes in the Pauli frame results in the following feedforward Boolean functions:

$$\begin{aligned}
\phi_1 &= m_3^{\mathrm{I}}(m_1^{\mathrm{I}} \oplus m_8^{\mathrm{I}} \oplus m_7^{\mathrm{I}}) \oplus m_1^{\mathrm{I}}(m_9^{\mathrm{I}} \oplus m_{10}^{\mathrm{I}}) \\
&\quad \oplus m_2^{\mathrm{I}} \oplus m_2^{\mathrm{P}} \oplus m_3^{\mathrm{P}} \oplus m_8^{\mathrm{I}} \oplus m_9^{\mathrm{I}} \oplus 1, \\
\phi_2 &= m^{\mathrm{A}} \oplus m_5^{\mathrm{I}} \oplus m_1^{\mathrm{P}}, \\
\phi_3 &= (m_1^{\mathrm{C}} \oplus m_5^{\mathrm{C}})(\phi_2 \oplus 1) \oplus m_6^{\mathrm{C}}\phi_2 \oplus m_4^{\mathrm{C}} \oplus m_5^{\mathrm{I}} \\
&\quad \oplus 1 \oplus m_1^{\mathrm{P}} \oplus m_6^{\mathrm{I}} \oplus m_{10}^{\mathrm{I}}(m_1^{\mathrm{I}} \oplus 1) \\
&\quad \oplus m_1^{\mathrm{I}}(m_9^{\mathrm{I}} \oplus 1 \oplus m_1^{\mathrm{P}}), \\
\phi_4 &= (m_2^{\mathrm{C}} \oplus m_6^{\mathrm{C}})(\phi_2 \oplus 1) \oplus m_5^{\mathrm{C}}\phi_2 \oplus m_3^{\mathrm{C}} \oplus m_4^{\mathrm{I}} \\
&\quad \oplus m_5^{\mathrm{I}} \oplus 1 \oplus m_1^{\mathrm{P}} \oplus m_7^{\mathrm{I}}(m_3^{\mathrm{I}} \oplus 1) \\
&\quad \oplus m_3^{\mathrm{I}}(m_8^{\mathrm{I}} \oplus 1 \oplus m_1^{\mathrm{P}}), \\
\phi_5 &= m^{\mathrm{A}} \oplus \phi_3 \oplus \phi_4,
\end{aligned} \tag{87}$$

where m_r^{P}, m_r^{I}, and m_r^{C} denote the preparation, injection, and correction outcomes, respectively, and m^{A} is the outcome of the Z measurement on b_k.

The first injection I_1 can be performed by initializing s_1 in $|+\rangle_L$, which encodes the value 0 in the color-changed convention, and computing $s_1 = a_1 \oplus b_1$ with three transversal Cnot gates $\mathrm{C}X_{b_1 a_1}$, $\mathrm{C}X_{s_1 b_1}$ and $\mathrm{C}X_{b_1 a_1}$. Alternatively, s_1 can be initialized in $|0\rangle_L$ and set to $a_1 \oplus b_1$ by an X-parity measurement on a_1, b_1 and s_1, with outcome m, followed by $Z_{s_1}^m$ in the Pauli frame. The last correction C_1 only measures $b_1 = a_1 \oplus s_1$ in the Z basis and applies $X_{a_1}^{m^{\mathrm{A}}} X_{s_1}^{m^{\mathrm{A}}}$ in the Pauli frame. In both cases, the first and last steps require no resource state.

C.5.2 Layout on the repetition-cat architecture

The Adder layout on the repetition-cat architecture, shown in Figure 43, is organized into five interacting zones. The *compute zone* stores the registers a, b and s of the Adder as interleaved repetition codes. The *buffer zone* is composed of neighboring *buffers*, each comprised of nine adjacent repetition codes of distance d, labeled $q_1, \ldots, q_9$. Each buffer retains an $|i\rangle$ catalyst at q_4 and can prepare either resource state. Delayed-$\mathrm{C}_X X$ preparation occupies only $q_2, \ldots, q_6$, leaving the other four columns idle. A buffer receives two $|i\rangle$ states from the factories for each preparation, together with a single $|\mathrm{C}_X \mathrm{C}_X X\rangle$ state for auto-And preparation, and prepares the corresponding resource state in place. The number of Adder buffers $n_{\mathrm{buf}}^{(\textsc{Adder})}$ is discussed in Section C.5.6 and the factory counts in Section C.4. Two distinct *routing zones* interface with the buffer zone. The upper compute routing zone spans both the compute and buffer zones, with four rows of data qubits. This zone supports the injection and correction phases in Figure 42 through simultaneous parity measurements. The *factory zone* consists of the $|\mathrm{C}_X \mathrm{C}_X X\rangle$ and $|i\rangle$ factories presented in Section C.4. The lower factory routing zone lies between the factories and the compute and buffer zones, with a height of $h_{\mathrm{routing,fac}}$ rows of data qubits and a length set by the factory placement. Similarly, this zone is reserved for routing factory outputs into the buffers via simultaneous parity measurements. In every zone, each row of data qubits is counted with the row of ancillae above it, i.e. two physical qubits per data qubit. An operation footprint is the sum of its zones. As such, the compute zone, the buffers and the upper routing require a core footprint

$$S_{\mathrm{core}}^{(\mathrm{rep})} = 2(d+4)(3n + 9n_{\mathrm{buf}}^{(\textsc{Adder})}), \tag{88}$$

where $S_{\mathrm{core}}^{(\mathrm{rep})}$ corresponds to the total number of physical qubits for the Adder, excluding factories and their associated routing. The factories are placed side by side along the factory routing zone. With n_{fac}^{j} factories of type $j \in \{|\mathrm{C}_X \mathrm{C}_X X\rangle, |i\rangle\}$, the factory zone spans a total length l_{fac} and requires S_{fac} physical qubits, given by

$$\begin{aligned}
l_{\mathrm{fac}} &= n_{\mathrm{fac}}^{|\mathrm{C}_X \mathrm{C}_X X\rangle}\, l_{\mathrm{fac}}^{|\mathrm{C}_X \mathrm{C}_X X\rangle} + n_{\mathrm{fac}}^{|i\rangle}\, l_{\mathrm{fac}}^{|i\rangle}, \\
S_{\mathrm{fac}} &= n_{\mathrm{fac}}^{|\mathrm{C}_X \mathrm{C}_X X\rangle}\, S_{\mathrm{fac}}^{|\mathrm{C}_X \mathrm{C}_X X\rangle} + n_{\mathrm{fac}}^{|i\rangle}\, S_{\mathrm{fac}}^{|i\rangle}.
\end{aligned} \tag{89}$$

The factory routing zone spans the same length and occupies

$$S_{\mathrm{routing,fac}} = 2h_{\mathrm{routing,fac}}\, l_{\mathrm{fac}} \tag{90}$$

physical qubits.

C.5.3 Layout on the LDPC-cat architecture

On the LDPC-cat architecture, the buffer, factories and their associated routing zones are those of the repetition-cat layout. The registers a, b and s are stored in a single open-boundary LDPC-cat code [16] of height H and width L, whose $2L$ logical qubits hold the $3n$ register bits, given $2L \geq 3n$. The code sits above the compute routing zone, and the first buffer lies under the last five of its L columns, with the other buffers to its right, as shown in Figure 47. The X-parity measurements of each injection and correction reach the register bits through their proxies (Section C.1.3), and the Z measurement of b_k through a repetition code covering the support of Z_L, as described in Section C.1.3 and shown in Figure 48. The compute zone of the repetition layer covers the footprint of the open-boundary code, including H rows and the proxy row over $p_\ell + L + p_r$ columns, where p_ℓ and p_r are the left and right edge paddings of the code. Each triple (a_k, b_k, s_k) occupies three consecutive columns, in increasing order of k. Row 1,

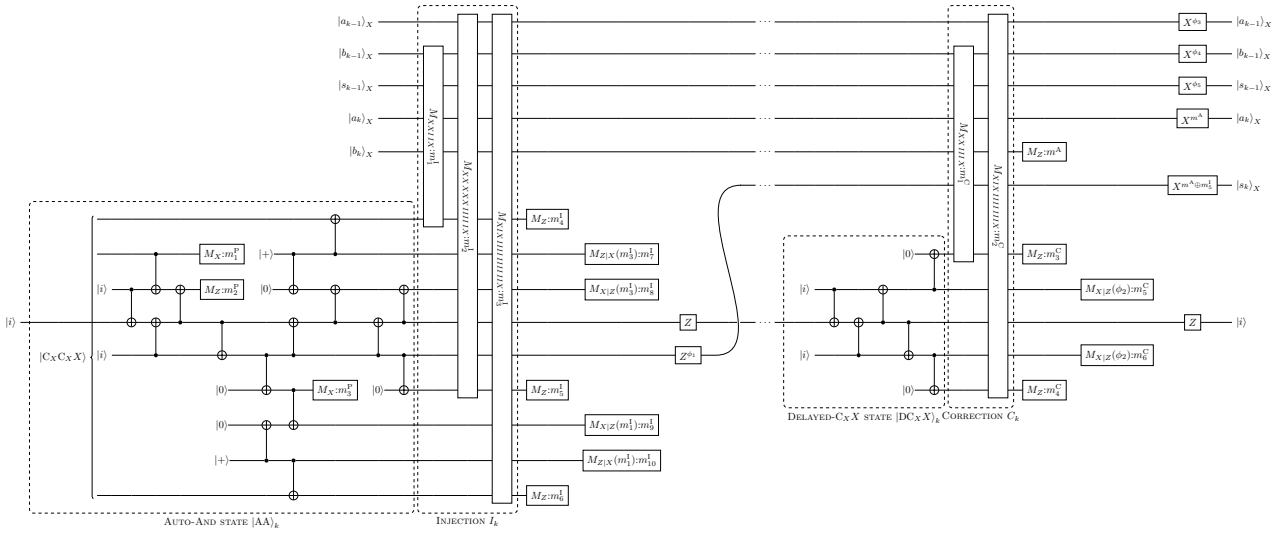


Figure 42: Logical circuit for the Adder iteration at step $k > 1$.

the bottom row, holds the first $\lceil n/2 \rceil$ triples, and Row 2, above it, the remaining ones. For odd n with $3\lceil n/2 \rceil > L_{\min}$, Row 1 holds the first $\lfloor n/2 \rfloor$ triples followed by a_n and b_n, such that s_n occupies the column of Row 2 above a_n. The proxy of s_n then lies in the first column of the right edge padding. As such, for an n-bit Adder, an LDPC code satisfying $L = \max(L_{\min}, \lceil 3n/2 \rceil)$ suffices. The LDPC layer holds the $HL + \Delta_{\mathrm{LDPC}}$ data qubits, where Δ_{LDPC} counts the qubits added by the open-boundary staircases, and $HL + \Delta_{\mathrm{LDPC}} - 2L$ ancillae. Therefore, the Adder on the LDPC-cat architecture, including the compute zone of both layers, the buffer zone and the compute routing zone, requires a core footprint of

$$\begin{aligned} S_{\mathrm{core}}^{(\mathrm{LDPC})} &= 2(H+1)(p_\ell + L + p_r) + 2(H-1)L \\ &\quad + 2\Delta_{\mathrm{LDPC}} + 18dn_{\mathrm{buf}}^{(\mathrm{Adder})} \\ &\quad + 8(L + 9n_{\mathrm{buf}}^{(\mathrm{Adder})} - 5). \end{aligned} \tag{91}$$

C.5.4 Resource state preparations

In the steady state, the Adder consumes an auto-And state and a delayed-$\mathrm{C}_X X$ state for the injection and correction steps, respectively. For inputs q_3, q_5 and a catalyst q_4, all initially in $|i\rangle$, up to Pauli frame updates, the four Cnot gates $\mathrm{C}X_{q_3 q_4}$, $\mathrm{C}X_{q_5 q_4}$, $\mathrm{C}X_{q_4 q_3}$ and $\mathrm{C}X_{q_4 q_5}$ produce the resource state $|\mathrm{C}Z\rangle = \mathrm{C}Z\,|+\rangle\,|+\rangle = \mathrm{C}_X X\,|0\rangle\,|0\rangle$. The auto-And preparation uses this state to teleport a Hadamard gate, and the delayed-$\mathrm{C}_X X$ state preparation performs a Z-copy of each $|\mathrm{C}Z\rangle$ qubit onto an auxiliary qubit, i.e. a Cnot from that qubit onto an auxiliary qubit initialized in $|0\rangle$. Each preparation therefore consumes two $|i\rangle$ states, while the catalyst is supplied once per buffer and per operation. The catalyst is seeded during the initialization of the operation and retained for all its preparations. Since the buffer zone is reconfigured between operations (Section C.7.1.ii), catalysts are not kept from one operation to the next.

i Auto-And state. The auto-And preparation receives the three $|\mathrm{C}_X\mathrm{C}_X X\rangle$ legs at q_1, q_2, q_9, two $|i\rangle$ states at q_3, q_5, and the recycled catalyst at q_4. It uses only local Cnot gates, measurements, and initializations (see Figure 42). First, the two $|i\rangle$ states and the catalyst form a $|\mathrm{C}Z\rangle$ state, which teleports a Hadamard gate from q_2 onto q_5. Then, two Bell pairs are prepared and entangled with the resulting $|\mathrm{C}_X\mathrm{C}_X Z\rangle$ state to implement delayed-choice Cnot corrections. These entangling operations are achieved through the remaining Cnot gates.

ii Delayed-$\mathrm{C}_X X$ state. A delayed-$\mathrm{C}_X X$ state is prepared from two $|i\rangle$ states at q_3, q_5, the catalyst at q_4, and two auxiliary qubits initialized in $|0\rangle$ at q_2, q_6. Assuming that the input states are ready, the preparation consists of six transversal Cnot gates, with independent operations performed in parallel.

The compiled preparation durations, shown in Figure 44, follow from Section C.1.1, such that $T_{\mathrm{prep}}^{|\mathrm{AA}\rangle} = 9$ and $T_{\mathrm{prep}}^{|\mathrm{DC}_X X\rangle} = 5$ cycles. The auto-And preparation contains the only X measurement of a $|0\rangle_L$ code made before any lattice surgery on it, m_3^{P} on q_6 (Section C.1.3). This does not lengthen $T_{\mathrm{prep}}^{|\mathrm{AA}\rangle}$, as I_k establishes the Pauli frame of m_3^{P} through the codes coupled to q_6, and m_3^{P} enters only ϕ_1 in Equation (87), which is evaluated after I_k. We omit Pauli corrections from the timing, as those are performed as a Pauli frame update in classical software with no associated delay on the state preparation.

iii Adaptive measurements. In both cases, the buffer remains occupied during the subsequent lattice surgery, throughout the reaction time and the final

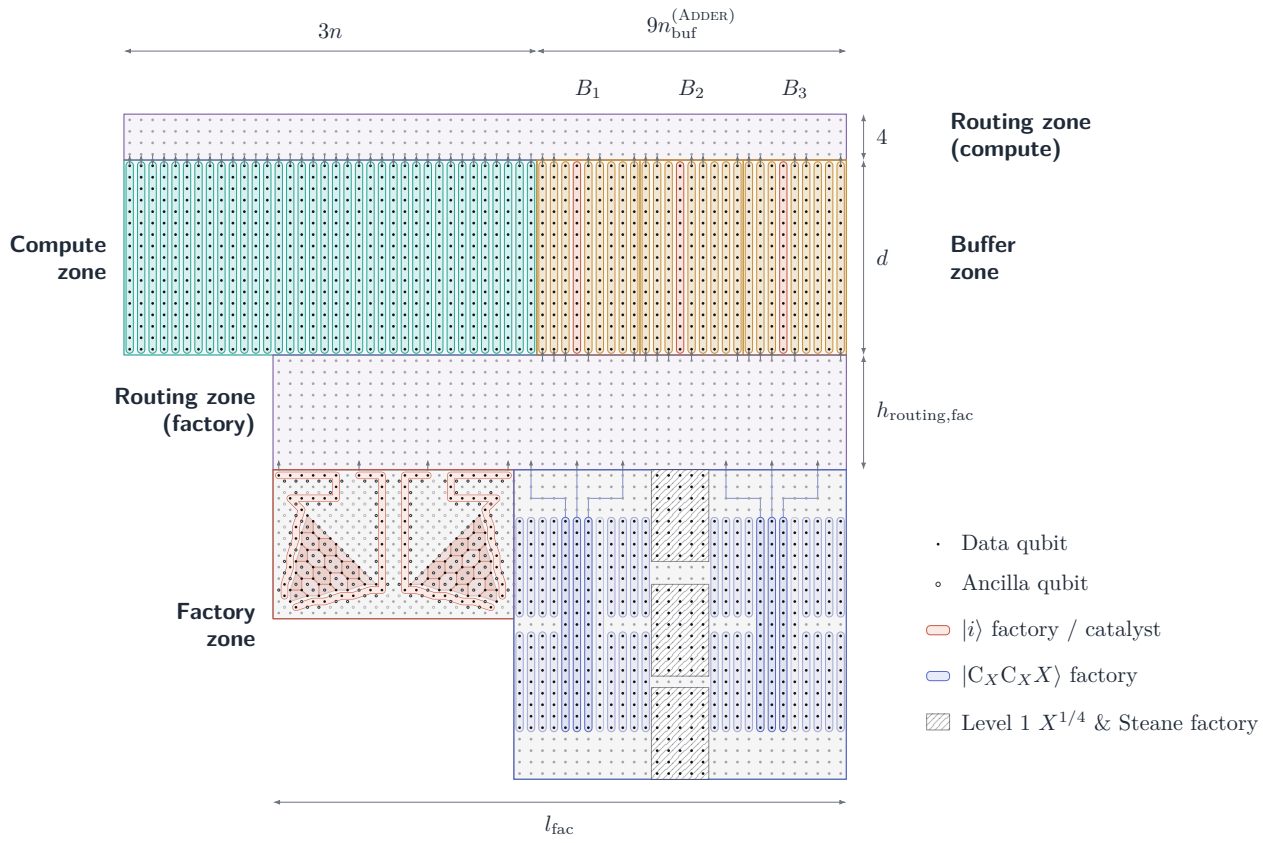


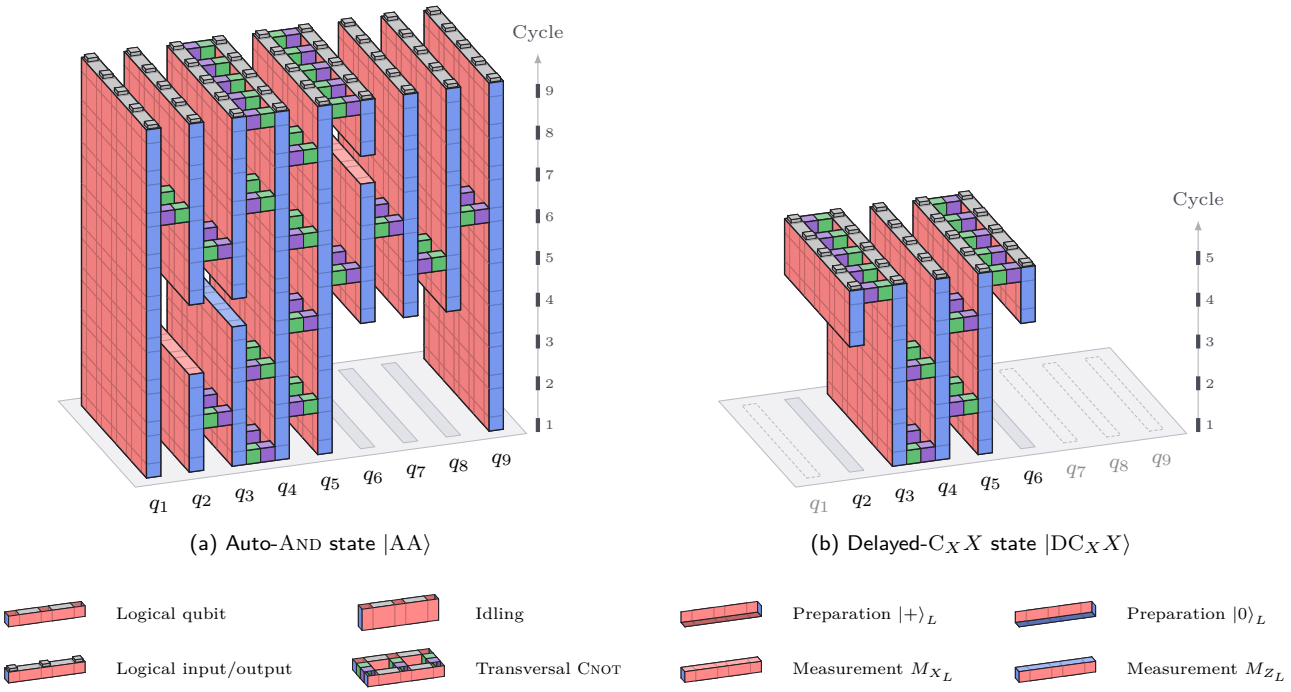
Figure 43: Illustrative layout of the repetition-cat Adder at $n = 12$, $d = 17$ and $T_{\text{cad}} = 21$. The compute zone stores the inputs and outputs (a_k, b_k, s_k) next to the three buffers B_1, B_2 and B_3. The compute routing zone above carries the simultaneous X-parity measurements of the injection and correction steps. The factory routing zone below carries factory outputs into the buffers. The factory zone holds two color code $|i\rangle$ factories and one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory, each with two output channels. Ancillae are drawn only in the color code factories, whose data and ancilla roles differ from the rest of the layout (Section C.1.1). Alternative layouts can be obtained by choosing different factory and buffer placements.

adaptive measurement. The final resource measurements therefore complete $T_{\text{react}} + T_{\text{meas}}$ after the lattice surgery.

iv Spacetime diagrams. The horizontal plane represents the physical layout, while the vertical axis represents time in cycles. Each cube represents a physical qubit. Two neighboring cubes are linked by a parallelepiped that represents an XX stabilizer. Red, blue, and gray faces mark X_L boundaries, Z_L boundaries, and logical interfaces, respectively. A vertical extension denotes idling through continued syndrome extraction. Red and blue lower caps prepare $|+\rangle_L$ and $|0\rangle_L$, respectively, while upper caps denote destructive X_L and Z_L measurements, respectively. A transversal Cnot gate is drawn as a connector with a green control and a purple target between adjacent tubes. Finally, the Pauli frame corrections are omitted. These conventions are presented in Figure 44.

C.5.5 Simultaneous lattice surgery

The injection and correction steps in Figure 42 involve three and two X-parity measurements, respectively. A possible implementation protocol consists of sequentially applying for each measurement a merge and split instruction between the repetition codes and the auxiliary qubits involved [4]. In the following, we instead use *simultaneous lattice surgery* to perform the measurements of each injection or correction step in a single lattice surgery [46]. For $k > 1$, the auto-And state consumed during injection I_k is injected into the codes associated with a_{k-1}, b_{k-1}, s_{k-1}, a_k, and b_k through their X_L logical representatives. Simultaneously, a routing of q_5 transfers the result into the slot reserved for s_k. The sum output $t = q_5$ is distinct from its Z-copy $t' = q_6$, which participates in one of the three injection measurements. Similarly, correction C_k for $k > 1$ applies the delayed-$\mathrm{C}_X X$ state through an X-parity measurement on b_{k-1}, s_{k-1}, and q_2, as well as another on a_{k-1}, s_{k-1}, and q_6. The stabilizer patterns shown in Figure 45 implement these simultaneous measurements and routing, with two complementary injection primitives and one correc-



(a) Auto-AND state $|\mathrm{AA}\rangle$

(b) Delayed-$\mathrm{C}_X X$ state $|\mathrm{DC}_X X\rangle$

Figure 44: Spacetime diagrams for the resource state preparations on the buffer layout $(q_1, \ldots, q_9)$ of the ADDER at $d = 5$. Cycles are spaced apart for readability.

tion primitive. In each case, the data qubits of the routing zone are initialized in $|0\rangle$, undergo d rounds of stabilizer measurements, and are measured destructively in the Z basis. The compute routing zone is released after the final Z measurements of each surgery. The consumed resource state continues to occupy its buffer until its reaction time and adaptive measurements are complete, while the catalyst is recycled for the next state preparation.

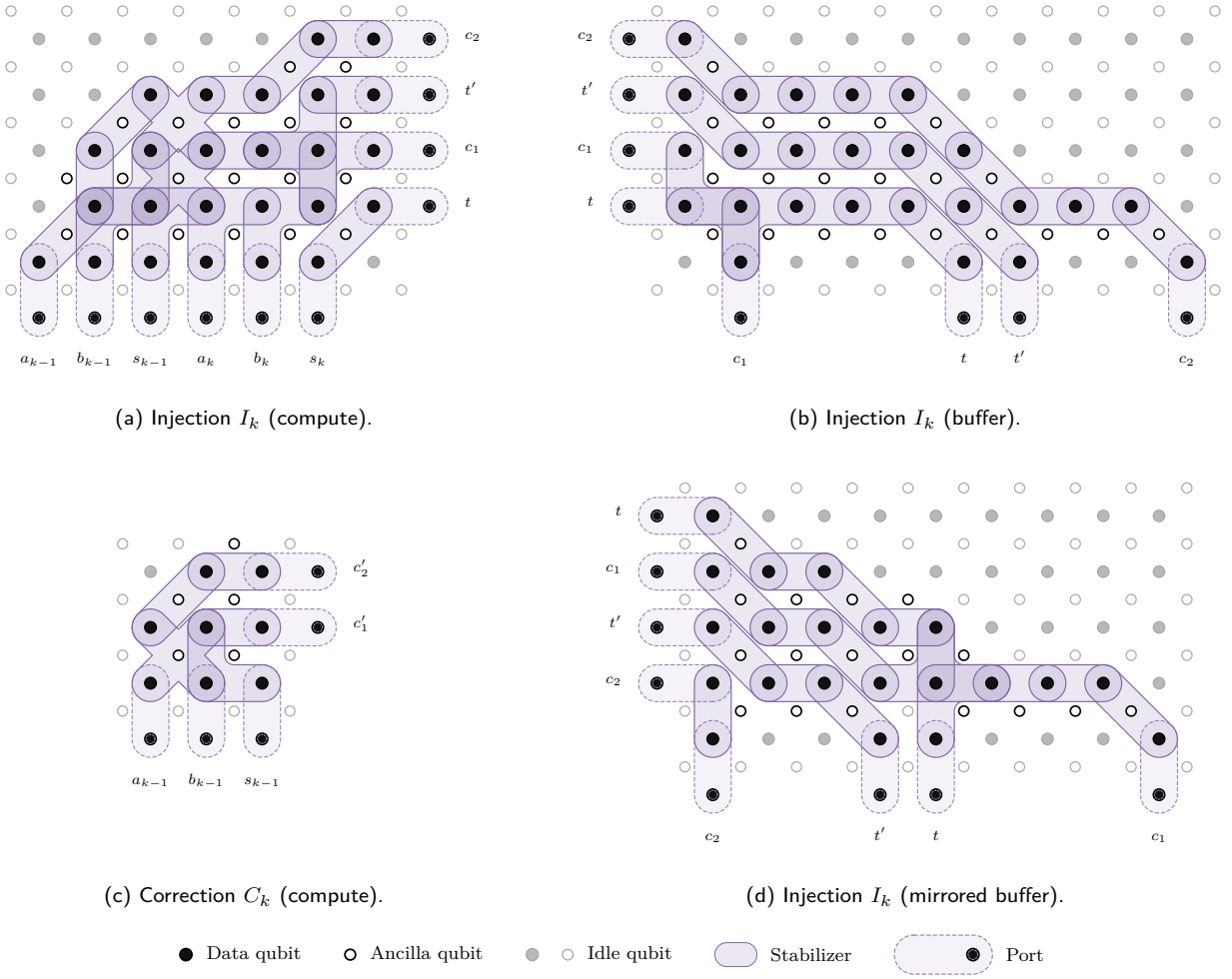
The injection primitive of Figure 45(a), whose horizontal arms end on the right, connects to the compute codes, and that of Figure 45(b), whose arms end on the left, connects to the prepared buffer codes. Their arms are joined in matching order. On both architectures, we embed these primitives into the four data rows of the compute routing zone, above the buffer zone.

Figure 46 shows the injection and correction layouts on the repetition-cat architecture. Figures 47 and 48 present the same steps on the LDPC-cat architecture, where the compute side uses the primitives of Figure 45(a) and (c) flipped top to bottom, and the buffers are mirrored left to right, such that the buffer side of the injection is the mirrored primitive of Figure 45(d). For the first buffer, which lies under the compute zone, an injection or correction associated with a compute tuple above can be achieved by combining the compute and buffer primitives into a single routing adjusted to the relative position. We note that this routing layout, designed for a compute zone on top of the buffers, can also be used for the repetition-cat architecture.

As for the injections and corrections, we allocate T_{surg} to each simultaneous loading of factory outputs into a buffer, during which the factory output and destination buffer are occupied. The factory routing zone of height $h_{\mathrm{routing,fac}}$ must accommodate five resource qubits for auto-AND preparation, comprising two $|i\rangle$ states and the three inputs of a single $|\mathrm{C}_X\mathrm{C}_X X\rangle$ state, or two fresh $|i\rangle$ states for delayed-$\mathrm{C}_X X$ preparation. As such, each buffer also serves as an $|i\rangle$ and $|\mathrm{C}_X\mathrm{C}_X X\rangle$ state storage area to maximize factory throughput.

C.5.6 Schedule

On the repetition-cat architecture, I_1 lasts $T_{\mathrm{init}} + 3T_{\mathrm{CNOT}}$, and C_1, a transversal Z measurement of b_1, is performed with the final Z measurements of the routing qubits of C_2, so that it adds no time. On the LDPC-cat architecture, assuming s_1 is initialized in $|0\rangle_L$, I_1 measures $X_{a_1}X_{b_1}X_{s_1}$ through the LDPC layer access zone, and C_1 measures the Z_L of b_1 in its own surgery after that of C_2, each lasting $T_{\mathrm{surg}}^{(\mathrm{LDPC})}$. For $k > 1$, the Z measurement of b_k is performed during the surgery of C_k on both architectures. After I_1, the $2(n-1)$ remaining steps start at the cadence T_{cad} of Equation (83), in the order $I_2, \ldots, I_n, C_n, \ldots, C_2$. Each surgery lasts T_{surg}, and the compute zone and the compute routing zone are idle for the remaining $T_{\mathrm{cad}} - T_{\mathrm{surg}}$ cycles. On the LDPC-cat architecture, these surgeries involve the LDPC layer, such that $T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ replaces T_{surg} in Equation (83) and in the following. Since the Pauli frame update ϕ_1 of I_k flips the outcomes m_1^{I} and m_3^{I} of I_{k+1}, which set



(a) Injection I_k (compute).

(b) Injection I_k (buffer).

(c) Correction C_k (compute).

(d) Injection I_k (mirrored buffer).

Figure 45: Simultaneous lattice surgery routing primitives of the injection and correction steps. Figures (a)–(c) are used on the repetition-cat architecture. On the LDPC-cat architecture, (a) and (c) are flipped top to bottom, and (d) replaces (b). In the injection, c_1 and c_2 are the control qubits, while t and t' are Z-copies of the target qubit from the auto-AND state. The qubit t is moved into s_k, and t' is involved in an X-parity measurement with the compute zone. In the correction, c'_1 and c'_2 are Z-copies of the control qubits c_1 and c_2 from the delayed-$\mathrm{C}_X X$ state.

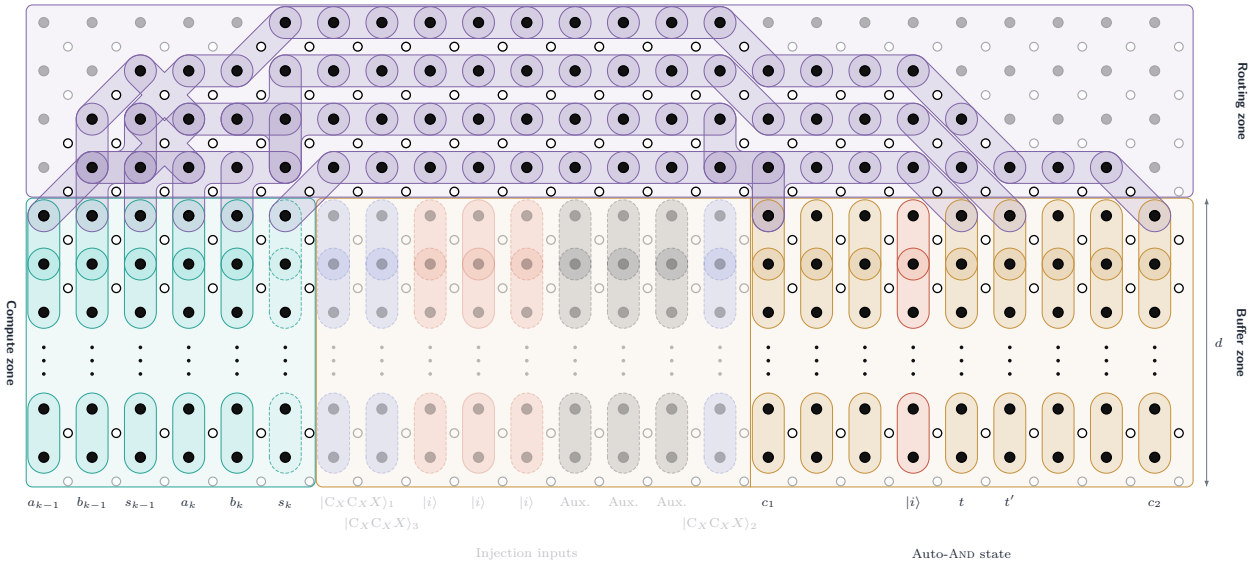
the bases of its adaptive measurements, the adaptive measurements of consecutive steps must be at least $T_{\mathrm{react}} + T_{\mathrm{meas}}$ apart, and similarly ϕ_4 of C_k flips m^{A} of C_{k-1}. This holds since $T_{\mathrm{react}} + T_{\mathrm{meas}} \leq T_{\mathrm{cad}}$.

i Buffers. Let $j \in \{|\mathrm{AA}\rangle, |\mathrm{DC}_X X\rangle\}$ denote the resource state consumed. For a step starting at t, loading begins at $t - T^j_{\mathrm{prep}} - T_{\mathrm{surg}}$ and preparation begins at $t - T^j_{\mathrm{prep}}$. A buffer remains reserved for T_{buf} cycles, through loading, preparation, surgery, and the final reaction and adaptive measurement. We assign the steps to the $n^{(\textsc{Adder})}_{\mathrm{buf}}$ buffers in round-robin order, with

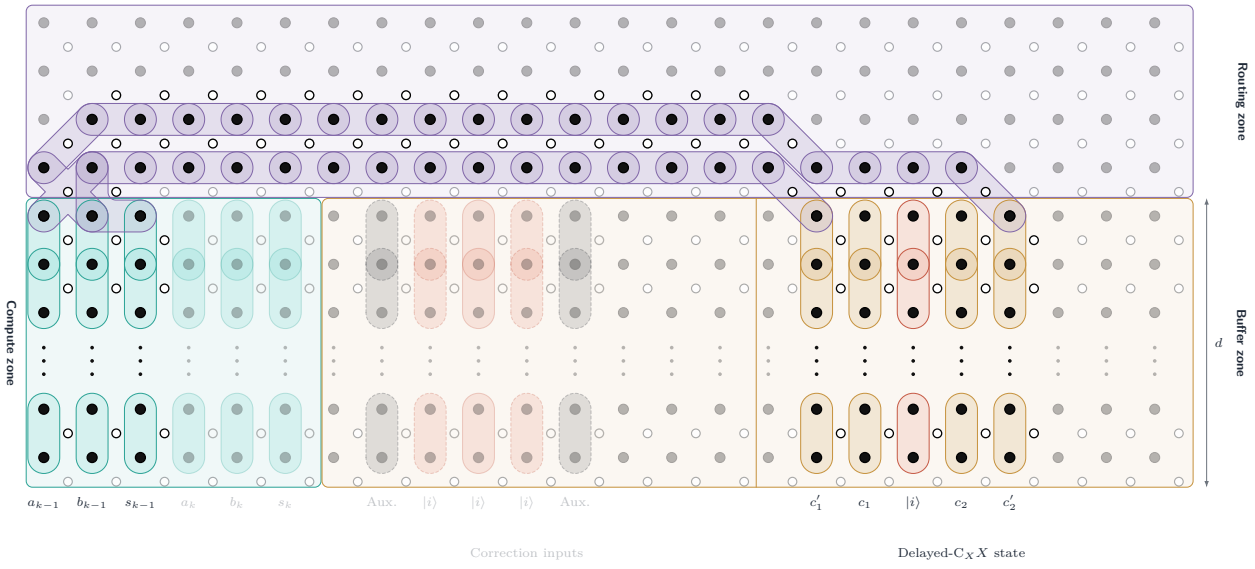
$$\begin{aligned} T_{\mathrm{buf}} &= 2T_{\mathrm{surg}} + \max\left(T^{|\mathrm{AA}\rangle}_{\mathrm{prep}}, T^{|\mathrm{DC}_X X\rangle}_{\mathrm{prep}}\right) \\ &\quad + T_{\mathrm{react}} + T_{\mathrm{meas}}, \\ n^{(\textsc{Adder})}_{\mathrm{buf}} &= \left\lceil \frac{T_{\mathrm{buf}}}{T_{\mathrm{cad}}} \right\rceil . \end{aligned} \tag{92}$$

Figure 49 shows the operation order and the complete buffer reservation.

ii Factories. During injection steps, one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ state is required every T_{cad}, correction steps require none, and every preparation requires two $|i\rangle$ states loaded simultaneously. The loading of a factory output state into a buffer is executed in parallel with the production of the following state. The full ADDER consumes $n-1$ $|\mathrm{C}_X\mathrm{C}_X X\rangle$ states and $4(n-1)$ $|i\rangle$ states, that is, four $|i\rangle$ states per Toffoli gate, two for the auto-AND state and two for the delayed-$\mathrm{C}_X X$ state. In addition, it consumes one catalyst seed per buffer, so that an n-bit ADDER consumes $4(n-1) + n^{(\textsc{Adder})}_{\mathrm{buf}}$ $|i\rangle$ states in total. The seeds are produced and loaded during initialization, and each catalyst is retained for the successive preparations of the subroutine.



(a) Injection I_k.



(b) Correction C_k.



Figure 46: Routing layout for an injection and a correction step ($k > 1$) on the repetition-cat architecture. Each figure contains a loading (left) and available (right) buffer.

iii Latency. The cadence of Equation (83) minimizes the number of factories. As the corrections $C_n, \ldots, C_2$ consume no $|\mathrm{C}_X\mathrm{C}_X X\rangle$ state, we note that the correction phase could run at a shorter cadence under the current numbers of factories and buffers, bounded only by the lattice surgery duration, the $|i\rangle$ production time and the buffer reservation time. For simplicity, we use the cadence T_{cad} for all steps, such that the latency of an n-bit ADDER is

$$T^{(\mathrm{rep})}_{\mathrm{ADDER}}(n) \simeq T^{(\mathrm{LDPC})}_{\mathrm{ADDER}}(n) \simeq 2nT_{\mathrm{cad}}. \tag{93}$$

We call this schedule *nominal*, as it assumes that every factory production completes within the associated mean time T^j_{prod}. A longer production can delay the step consuming its output, and with it all later steps. This nominal latency is measured from the start of I_1 and excludes initialization.

C.5.7 Summary

Adding the core, factories and their associated routing results in a total footprint of

$$S_{\mathrm{total}} = S_{\mathrm{core}} + S_{\mathrm{fac}} + S_{\mathrm{routing,fac}}, \tag{94}$$

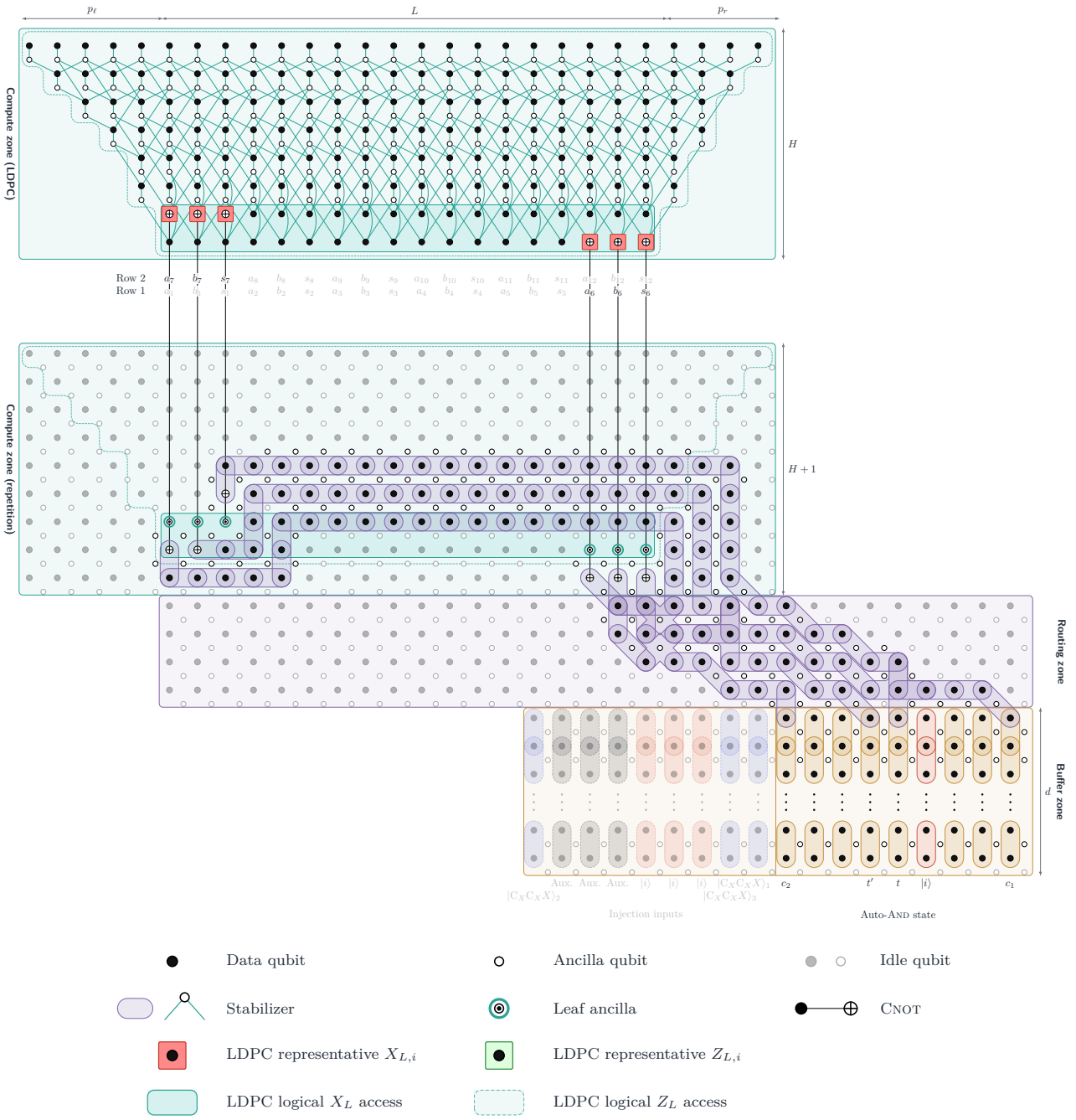


Figure 47: Layout for the injection step I_7 of the ADDER on the LDPC-cat architecture for $n = 12$ and $d_{\text{LDPC}} = 22$. The compute zone lies above the routing zone and the buffers are mirrored left to right.

with $S_{\text{core}} = S_{\text{core}}^{(\text{rep})}$ or $S_{\text{core}}^{(\text{LDPC})}$ chosen for the repetition-cat and LDPC-cat architectures, respectively.

The estimates use the factory parameters of Table 19 and the operation durations of Section C.1.1, such that $T_{\text{buf}} = 2d + 20$ in Equation (92).

We choose $h_{\text{routing,fac}} = 10$ as a margin of two data rows for each of the five resource qubits in an injection load.

Table 21 provides the footprint of an n-bit ADDER under various distances, where the LDPC-cat registers are stored in the code of Table 20 whose distance d_{LDPC} is the smallest one at least d_{rep}. The choice of T_{cad} from Equation (83) gives a latency of about $42n$ cycles at $d_{\text{rep}} = 17$, or about $44n$ cycles on the LDPC-cat architecture with $d_{\text{LDPC}} = 22$. Under this cadence choice, one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory and two $|i\rangle$ factories suffice at every distance (Section C.4). The factory overhead, including routing, is $S_{\text{routing,fac}} + S_{\text{fac}} = 2872$ qubits. Figure 49 illustrates the complete schedule at $n = 12$ for this parameter choice.

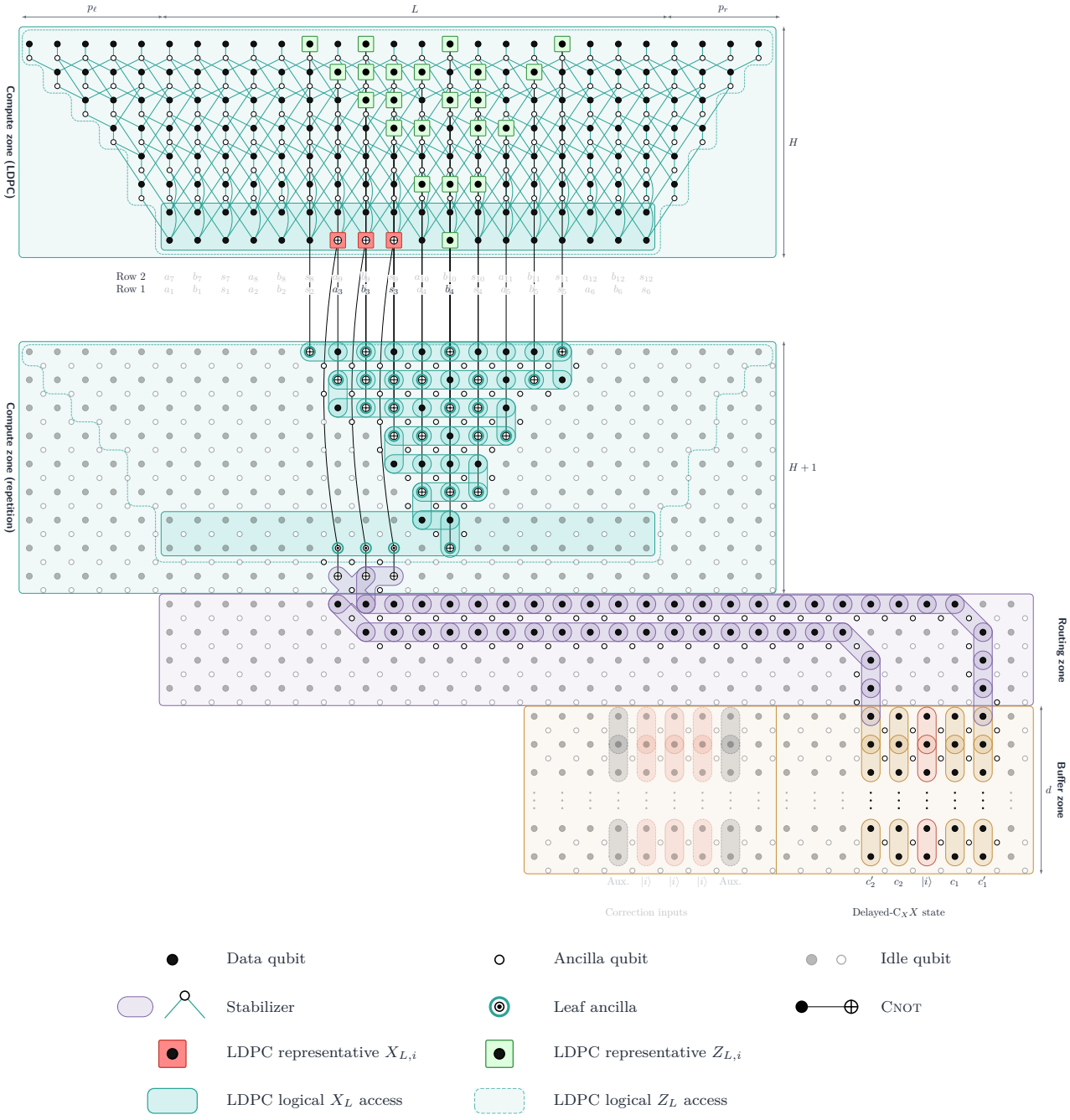


Figure 48: Layout for the correction step C_4 of the ADDER on the LDPC-cat architecture, similarly to Figure 47.

C.6 LOOKUP and PHASEUP

i Lookup A LOOKUP operation, also called a QROM operation (quantum read-only memory), is an operation that loads values from a classical table in a quantum target register using an address register. Let w be the size of the address register, n the size of the target register and t the classical table that maps each address to an n-bit value:

$$\begin{aligned} t : \{0,1\}^w &\longrightarrow \{0,1\}^n \\ k &\longmapsto t(k) \end{aligned} \tag{95}$$

Let $b \in \{0,1\}^n$ and $(a_k)_{k\in[0,2^w-1]}$ such that $\sum_{k=0}^{2^w-1} |a_k|^2 = 1$. The LOOKUP associated with t acts as:

$$\text{LOOKUP} \sum_{k=0}^{2^w-1} a_k |k\rangle |b\rangle = \sum_{k=0}^{2^w-1} a_k |k\rangle |b \oplus t(k)\rangle . \tag{96}$$

In this work, we use the cat-qubit implementation of the skew-tree QROM detailed in [47]. We recall the duration and the number of repetition-code columns derived in [47] for the reader's convenience. With the repetition-cat architecture, the duration and the number of repetition-code columns of this implementation,

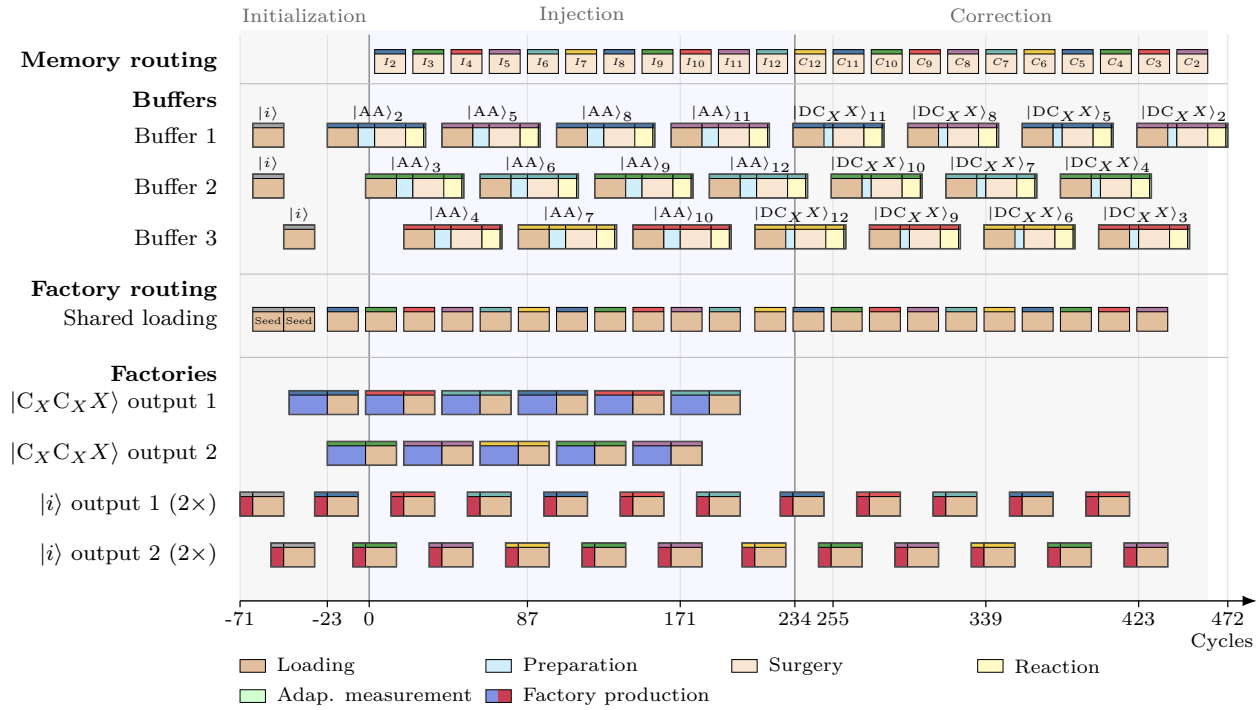


Figure 49: Complete nominal schedule of a repetition-cat ADDER at $n = 12$, $d = 17$ and $T_{\text{cad}} = 21$.

Code	H	$L_{\min}$	p_ℓ	p_r	Δ_{LDPC}	LDPC layer
$[8L + 29, 2L, 22]$	8	17	5	4	29	$14L + 58$
$[7L + 26, 2L, 16]$	7	17	4	5	26	$12L + 52$
$[6L + 17, 2L, 12]$	6	13	3	4	17	$10L + 34$

Table 20: Open-boundary LDPC-cat codes of height H and width L [16], with $L = \max(L_{\min}, \lceil 3n/2 \rceil)$ for an n-bit ADDER. The paddings p_ℓ and p_r extend the code to the left and right, Δ_{LDPC} counts the qubits added by the open-boundary staircases, and the last column provides the number of data and ancilla qubits in the LDPC layer.

for a w-qubit address register and an n-qubit target register, are

$$\begin{aligned} T^{(\text{rep})}_{\text{LOOKUP}}(w, d) &= (2^w - 1)\, d + (2^{w-1} - w)\, T_{\text{cad}}, \\ n^{(\text{rep})}_{\text{col,LOOKUP}}(w, n) &= 2w + 3\lceil n/2 \rceil + 6\, n^{(\text{LOOKUP})}_{\text{buf}}, \end{aligned} \tag{97}$$

with $T^{(\text{rep})}_{\text{LOOKUP}}$ the duration of the LOOKUP, $n^{(\text{rep})}_{\text{col,LOOKUP}}$ the number of repetition-code columns of its footprint, including its own routing columns, d the distance of the repetition code, T_{cad} the cadence of Equation (83) and $n^{(\text{LOOKUP})}_{\text{buf}}$ the number of LOOKUP buffers. In reverse depth-first order, the skew-tree QROM consists of $2^w - 1$ parity-CNOT gates and $2^w - w - 1$ AND gates [47]. The $2^{w-1} - 1$ AND gates that are directly followed by a parity-CNOT gate are performed simultaneously with it. The two gates use disjoint routing paths. The AND gate and the parity-CNOT gate performed simultaneously are then followed by a measurement-based AND uncomputation and another parity-CNOT gate, which leaves the factory $2T_{\text{surg}}$ to produce the next $|C_XC_XX\rangle$ state. Only the $2^{w-1} - w$ AND gates followed by another AND gate wait for the cadence. This requires $t_{|C_XC_XX\rangle} \leq 2T_{\text{surg}}$, i.e. $d \geq 11$ with the factories of Table 19, which holds at every distance considered in this work [47, Eq. (4.8)]. A LOOKUP buffer [47] consists of six adjacent repetition-code columns, in which the $|C_XC_XX\rangle$ and $|i\rangle$ resource states consumed by the AND gates of the LOOKUP are prepared. It differs from the nine-column ADDER buffer of Section C.5. Three of these columns receive the $|C_XC_XX\rangle$ state, and the other three are $|i\rangle$ columns, one of which holds a catalyst, as in the ADDER buffer (Section C.5.4). Each AND gate thus consumes one $|C_XC_XX\rangle$ state and two $|i\rangle$ states. Including one catalyst seed per buffer, loaded during the initialization of the LOOKUP, a LOOKUP consumes $2(2^w - w - 1) + n^{(\text{LOOKUP})}_{\text{buf}}$ $|i\rangle$ states. The auxiliary qubits of the LOOKUP are uncomputed by measurement-based uncomputation [90]. The resulting phase corrections are C_XX gates, that is, signs depending only on the address register. They are not applied in the LOOKUP but folded into the classical table of the PHASEUP that resolves its phases, under the same condition as for combining LOOKUPs: the address register is used only as a control in between. They therefore consume no resource state and add no

		Repetition-cat		LDPC-cat	
d_{rep}	d_{LDPC}	$n_{\mathrm{buf}}^{(\textsc{Adder})}$	$S_{\mathrm{core}}^{(\mathrm{rep})}$	$n_{\mathrm{buf}}^{(\textsc{Adder})}$	$S_{\mathrm{core}}^{(\mathrm{LDPC})}$
11	12	2	$90n + 540$	3	$32L + 902$
13	16	3	$102n + 918$	3	$36L + 1074$
16	16	3	$120n + 1080$	3	$36L + 1236$
17	22	3	$126n + 1134$	3	$40L + 1314$

Table 21: Footprint without factories, in qubits, of an n-bit Adder for the cadence of Equation (83) and the operation durations of Section C.1.1. $L = \max(L_{\min}, \lceil 3n/2 \rceil)$ is the width of the LDPC code of Table 20.

step to Equation (97). The Lookup buffers therefore account for $6n_{\mathrm{buf}}^{(\textsc{Lookup})}$ repetition-code columns in Equations (97) and (98). Each repetition-code column holds $2d$ physical qubits (Section C.5), and the routing rows used by the Lookup are counted in the global routing (Section C.7.1). For the LDPC-cat architecture, the duration and the number of repetition-code columns are

$$\begin{aligned} T_{\textsc{Lookup}}^{(\mathrm{LDPC})}(w) &= (2^w - 1)\, T_{\mathrm{surg}}^{(\mathrm{LDPC})} + (2^{w-1} - w)\, T_{\mathrm{cad}}, \\ n_{\mathrm{col},\textsc{Lookup}}^{(\mathrm{LDPC})}(w, n_{\mathrm{buf}}^{(\textsc{Lookup})}) &= 6\, n_{\mathrm{buf}}^{(\textsc{Lookup})} + w - 1, \end{aligned} \tag{98}$$

with $T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ the surgery duration of Equation (78), so that the duration is that of Equation (97) with d replaced by $T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ [47, Eq. (4.12)]. The address and target registers are stored in the LDPC layer, so that $n_{\mathrm{col},\textsc{Lookup}}^{(\mathrm{LDPC})}$ counts only the repetition-code columns of its buffers and of its $w - 1$ auxiliary qubits, in the repetition layer [47]. All Lookups in this paper, use $n_{\mathrm{buf}}^{(\textsc{Lookup})} = 2$.

ii Phaseup A Phaseup on a w-qubit address register applies a phase that depends only on the value of this register, $\sum_k a_k \left|k\right\rangle_X \mapsto \sum_k (-1)^{T_k} a_k \left|k\right\rangle_X$, where $T : \{0,1\}^w \to \{0,1\}$ is a classically known table. In this work, Phaseups resolve the phases left by the measurement-based uncomputation [90] of Lookups. Following the color-changed convention introduced in Section C.3, measurement-based uncomputation clears the target register of the Lookup by measuring each target qubit in the Z basis. This measurement leaves on the address register, whose states are written in the X basis, a sign $(-1)^{T_k}$, where the table T depends on the measurement outcomes. A Lookup is therefore uncomputed by this measurement followed by a Phaseup that removes this sign. It does not need to be applied immediately after the measurement: it can be delayed as long as the address register is only used as control qubits between the measurement and the Phaseup. Under this condition, the corrections of several Lookups that share the same address register can be combined into a single Phaseup.

We use the color-changed version of the Phaseup construction of [11, Appendix A.3]. The address register is split into a high half of $h = \lceil w/2 \rceil$ qubits and a low half of $l = \lfloor w/2 \rfloor$ qubits, each expanded into its power product with And gates. The empty product of each half is omitted, and the single-qubit entries are the address qubits themselves, so only the $2^h - h - 1$ and $2^l - l - 1$ entries of weight at least two require an And gate. The sign is then applied by multi-target $\mathrm{C}_X X$ gates, and the power products are uncomputed by measurement-based uncomputation. Counting the gates of this construction, a Phaseup on a w-qubit address register requires $2^h + 2^l - w - 2$ And gates, $2^l - 1$ multi-target $\mathrm{C}_X X$ gates and $2^h + 2^l - 2$ workspace qubits, including the address register. This agrees with the leading-order count of [11] and, for $w = 6$, with the 8 Toffoli gates quoted there. On the repetition-cat architecture, the $2^h + 2^l - w - 2$ And gates are cadence-limited and each of the $2^l - 1$ multi-target $\mathrm{C}_X X$ gates takes one lattice surgery, so that this logical cost translates into the following approximate physical cost:

$$\begin{aligned} T_{\textsc{Phaseup}}^{(\mathrm{rep})}(w, d) &\simeq \left(2^h + 2^l - w - 2\right) T_{\mathrm{cad}} \\ &\quad + \left(2^l - 1\right) d, \\ n_{\mathrm{col},\textsc{Phaseup}}^{(\mathrm{rep})}(w, n_{\mathrm{buf}}^{(\textsc{Lookup})}) &= 2^h + 2^l - 2 + 6\, n_{\mathrm{buf}}^{(\textsc{Lookup})}. \end{aligned} \tag{99}$$

with $T_{\textsc{Phaseup}}^{(\mathrm{rep})}$ the duration of the Phaseup, $n_{\mathrm{col},\textsc{Phaseup}}^{(\mathrm{rep})}$ the number of repetition-code columns of its workspace and buffers, d the distance of the repetition code, and $n_{\mathrm{buf}}^{(\textsc{Lookup})} = 2$ the number of Lookup buffers, each made of six repetition-code columns (see [47]). We neglect the time contribution of the measurement-based uncomputation of the And gates. The And gates of the Phaseup consume the same resource states as the And gates of a Lookup. Each requires one $\left|\mathrm{C}_X \mathrm{C}_X X\right\rangle$ state and two $\left|i\right\rangle$ states loaded into a buffer, and is issued at the cadence T_{cad} of Equation (83). A multi-target $\mathrm{C}_X X$ gate with control x and targets $y_1, \ldots, y_m$ applies the phase $(-1)^{x \cdot (y_1 \oplus \cdots \oplus y_m)}$, that is, a single $\mathrm{C}_X X$ gate between x and the X-parity of its targets. Since $\mathrm{C}_X X$ maps Z_x to $Z_x X_y$, it cannot be implemented with Cnot gates and X- and Z-basis preparations and measurements alone. We apply it with a $\left|\mathrm{C}Z\right\rangle = \mathrm{C}Z \left|+\right\rangle \left|+\right\rangle$ state, prepared in a Lookup buffer from two $\left|i\right\rangle$ states and the catalyst with the four Cnot gates of Section C.5.4. One lattice surgery that measures the X-parity of each $\left|\mathrm{C}Z\right\rangle$ qubit with the corresponding data, followed by Z measurements of the two $\left|\mathrm{C}Z\right\rangle$

qubits. The gate thus consumes no $|\mathrm{C}_X\mathrm{C}_X X\rangle$ state. Unlike the corrections of the ADDER, the $\mathrm{C}_X X$ gates of the PHASEUP do not require a delayed-$\mathrm{C}_X X$ state: whether each of them is applied is known before its surgery, so no delayed choice, and hence no Z-copy, is needed. The measurement-based uncomputation of a power product requires, with probability $1/2$, a $\mathrm{C}_X X$ phase correction between its two factors. We apply these phase corrections adaptively: once the measurement outcome is known, after the reaction time, a $|\mathrm{C}Z\rangle$ state is used only if the phase correction is needed, and otherwise kept in its buffer for the next phase correction. This consumes on average one $|i\rangle$ state per power product. Including the two catalyst seeds of the LOOKUP buffers, a PHASEUP on a w-qubit address register consumes on average

$$N_{|i\rangle}^{\text{PHASEUP}}(w) = 3\left(2^h + 2^l - w - 2\right) + 2\left(2^l - 1\right) + 2 \tag{100}$$

$|i\rangle$ states, that is 40 for $w = 6$ and 23 for $w = 5$. Each AND gate occupies a buffer for the same time as an AND gate of a LOOKUP, and the AND gates are issued at the same rate. The PHASEUP thus needs the same number of buffers, $n_{\text{buf}}^{(\text{LOOKUP})} = 2$, which adds $6n_{\text{buf}}^{(\text{LOOKUP})}$ repetition-code columns to its footprint. A PHASEUP never runs concurrently with a LOOKUP, so in the global layout it reuses the LOOKUP buffers and adds no qubits (Section C.7.1). As everywhere, a repetition-code column costs $2d$ physical qubits, and routing qubits are counted in the global layout. Equation (99) neglects the duration of the measurement-based uncomputation of the power products, including the reaction time and the surgeries of the adaptive phase corrections. The equation also assumes that the multi-target $\mathrm{C}_X X$ gates are issued every d cycles, whereas each of them now also occupies a buffer with a $|\mathrm{C}Z\rangle$ state. Approximate estimates in space and time are acceptable for PHASEUP operations since they are not determinant for global layouts and represent a small fraction of the algorithm's total duration (see Table 25). On the LDPC-cat architecture, the address register is stored in the LDPC layer, whereas the power-product auxiliaries and the LOOKUP buffers are stored in the repetition layer. The cost of Equation (99) then becomes

$$\begin{aligned} &T_{\text{PHASEUP}}^{(\text{LDPC})}(w, d_{\text{LDPC}}) \\ &\quad \simeq \left(2^h + 2^l - w - 2\right) T_{\text{cad}} + \left(2^l - 1\right) d_{\text{LDPC}}, \\ &n_{\text{col,PHASEUP}}^{(\text{LDPC})}(w, n_{\text{buf}}^{(\text{LOOKUP})}) \\ &\quad = 2^h - h + 2^l - l - 2 + 6\, n_{\text{buf}}^{(\text{LOOKUP})}, \end{aligned} \tag{101}$$

The duration is that of Equation (99) with d replaced by d_{LDPC} for the multi-target $\mathrm{C}_X X$ gates. Compared with Equation (99), the h high and l low address qubits, which are the single-qubit entries of the two power products, are now in the LDPC layer and leave the repetition-code workspace. This workspace thus reduces to the $2^h - h - 1$ and $2^l - l - 1$ entries of weight at least two, that is, one repetition-code column per AND gate, plus the $6n_{\text{buf}}^{(\text{LOOKUP})}$ columns of the buffers. As for Equation (99), routing qubits are not included, and the approximations listed above apply.

C.7 Repetition-cat architecture

In this section, we estimate the physical resources required to factor a 2048-bit RSA integer on the repetition-cat architecture. Section C.7.1 describes the layout, the duration and the volume of the computation for the algorithmic parameters of Table 22 at an arbitrary code distance d. Section C.7.2 then evaluates the estimate at $d = 13$, 15 and 17 for $p_Z = 10^{-3}$ and $p_Z = 5 \times 10^{-4}$.

C.7.1 Layout, duration and volume at distance d

i Introduction This section estimates the number of physical qubits, the duration and the space-time volume of one shot of the algorithm of Appendix B on the repetition-cat architecture, as functions of the code distance d. The estimate uses the physical implementation of the operations of Sections C.5 and C.6 and the algorithmic parameters of Table 22. Following Table 15, we divide the algorithm into temporal slices, and for each slice we count the repetition-code columns it uses and its duration. We assume an error-correction cycle time of $1\,\mu$s, a reaction time of $10\,\mu$s, and no bit flip during the computation. The factories are those of Section C.4: one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory and two $|i\rangle$ factories, shared by all operations, which run at the cadence $T_{\text{cad}} = \max(d, 21)$ of Equation (83). We fix the number of ADDER buffers to $n_{\text{buf}}^{(\text{ADDER})} = 3$. This satisfies Equation (92) for every d, since $T_{\text{buf}} = 2d + 20 \leq 3\max(d, 21)$, and it is the minimum for $d \geq 12$. The sufficient condition $T_{\text{react}} + T_{\text{meas}} = 11 \leq T_{\text{cad}}$ also holds. LOOKUPs and PHASEUPs use $n_{\text{buf}}^{(\text{LOOKUP})} = 2$ resource-state buffers of six repetition codes each (Section C.6). Physical qubits are counted in two parts. Every logical qubit, resource state or operation slot is held in a repetition-code column: d data qubits, each counted with the ancilla directly above it, i.e. $2d$ physical qubits (Section C.5). The active columns of a LOOKUP are its $n_{\text{col,LOOKUP}}^{(\text{rep})}$ columns (Equation (97)), which include its own routing columns. All other qubits of the chip (except factory qubits) are routing qubits: the routing rows, the routing column of the compute zone and the factory routing zone. They are never counted within an operation and instead form the global routing Γ_{global}, which is fixed by the global layout and present during the whole shot. A slice s therefore costs $\Gamma_{\text{physical}}(s) = 2d\, n_{\text{col}}(s)$ physical qubits, with $n_{\text{col}}(s)$ its number of repetition-code columns (Table 23), on top of Γ_{global} and of the factories. The factories enter only through their footprint n_{fac} and the factory routing zone, of height

$h_{\text{routing,fac}}$ and length l_{fac}, whose values are given in Section C.7.2.

ii Layout Figure 50 shows the global layout. The arrangement of the zones is set by the most frequent operations, LOOKUP and ADDER, and the length of the chip by the widest slice of Table 23. The chip consists of two rectangles stacked vertically, each one repetition code high, with a routing zone between them that spans the length of the chip. The top rectangle and the left part of the bottom one store the exponent. The right end of the bottom rectangle holds the buffer zone, which stores resource states. The compute zone, which consumes them, sits directly to its left, and the memory zone, between the bottom exponent zone and the compute zone, stores the remaining logical qubits. The routing zone has four rows above the memory, compute and buffer zones. Above the memory and compute zones, these rows provide the three lanes and the crossing row used to move registers [46], and above the buffers they carry the injection and correction routing of the ADDER (Section C.5.5). During a LOOKUP, the lowest of these rows holds the $|0\rangle_L$ code of the parity-CNOT gates and the three others carry the X-parity measurements of the AND gates (see below). Two rows suffice between the two exponent zones, since exponent qubits are only used as LOOKUP addresses and controls, so that only their X_L operator needs to be accessible. In the color-changed implementation (Section C.3), this also holds for the inverse quantum Fourier transform at the end of the algorithm, which needs only X_L access to the exponent qubits and a transversal destructive Z-basis measurement. Exponent qubits are therefore stored side by side, with no routing between them. The compute zone holds the $3(\kappa+1)$ interleaved slots of the ADDER† of ACCUMULATERESIDUE, plus one routing column on its left, which gives the LOOKUP Z_L access to its leftmost output, hence $l_{\text{comp}} = 3(\kappa+1)+1$. The memory zone holds two ℓ-qubit registers and the quotient register, $l_{\text{mem}} = 2\ell + \log(n_e)$, and the buffer zone is sized for the ADDER, $l_{\text{buf}} = 9n_{\text{buf}}^{(\text{ADDER})}$. The factories sit below the buffer zone, separated from it by $h_{\text{routing,fac}}$ rows that route resource states to the buffers.

Lengths are counted in repetition-code columns. Both rectangles have the same length, so that the chip has length

$$l_{\text{tot}} = \left\lceil \frac{l_{\text{buf}} + l_{\text{comp}} + l_{\text{mem}} + n_e}{2} \right\rceil, \tag{102}$$

and the bottom exponent zone has length $l_{\text{exp},1} = l_{\text{tot}} - (l_{\text{buf}} + l_{\text{comp}} + l_{\text{mem}})$, the top rectangle holding the remaining exponent qubits. The chip thus offers $2l_{\text{tot}}$ repetition-code columns. Each row of data qubits owns the row of ancillae above it, so that a zone of h rows of data qubits and l columns costs $2hl$ physical qubits. The global layout adds the zones and subtracts the ancilla rows above exposed zone tops, i.e. zone tops with nothing above them, since an ancilla row is only needed between two zones. Here, the only exposed tops are those of the top rectangle, with l_{tot} ancillae. Both routing zones lie under the top rectangle. The factory routing zone lies under the buffer zone and, for its $l_{\text{fac}} - l_{\text{buf}}$ other columns, under the compute zone, so that it is not exposed either. The global routing consists of the factory routing zone, the routing zone between the two exponent zones, the routing zone above the memory, compute and buffer zones, and the routing column of the compute zone:

$$\begin{aligned}\Gamma_{\text{global}} = {} & 2h_{\text{routing,fac}}\, l_{\text{fac}} + 2 \times 2\, l_{\text{exp},1} \\ & + 2 \times 4\,(l_{\text{buf}} + l_{\text{comp}} + l_{\text{mem}}) + 2d.\end{aligned} \tag{103}$$

The total number of physical qubits is the sum of the zones minus the exposed tops,

$$\begin{aligned}n_{\text{total}} = {} & 2d \cdot 2l_{\text{tot}} + 2 \times 2\, l_{\text{exp},1} \\ & + 2 \times 4\,(l_{\text{buf}} + l_{\text{comp}} + l_{\text{mem}}) \\ & + 2h_{\text{routing,fac}}\, l_{\text{fac}} - l_{\text{tot}} + n_{\text{fac}}.\end{aligned} \tag{104}$$

For RSA-2048, the parameters of Table 22 give $l_{\text{comp}} = 103$, $l_{\text{mem}} = 51$ and $l_{\text{buf}} = 27$. The chip has length $l_{\text{tot}} = 731$ and the bottom exponent zone has length $l_{\text{exp},1} = 550$. Since $l_{\text{buf}} + l_{\text{comp}} + l_{\text{mem}} + n_e = 1461$ is odd, the top rectangle holds the remaining 730 exponent qubits in 731 columns, so one exponent slot is unused. Equations (103) and (104) become

$$\Gamma_{\text{global}}(d) = 2d + 3648 + 2h_{\text{routing,fac}}\, l_{\text{fac}}, \tag{105}$$

$$\begin{aligned}n_{\text{total}}(d) = {} & 2924\, d + 3648 + 2h_{\text{routing,fac}}\, l_{\text{fac}} \\ & - 731 + n_{\text{fac}},\end{aligned} \tag{106}$$

where the constant of $\Gamma_{\text{global}}(d)$ collects the routing zone between the exponent zones (2200) and the routing zone above the memory, compute and buffer zones (1448), and the routing column gives $2d$. In $n_{\text{total}}(d)$, 3648 is the routing of $\Gamma_{\text{global}}(d)$ outside the routing column, and $731 = l_{\text{tot}}$ removes the ancillae above the exposed top of the top rectangle. The column counts $n_{\text{col}}(s)$ of Table 23 do not depend on d. Their maximum, 1460, is reached by the Q-Q ADDER$^\dagger(\kappa+1)$ and the Q-Q COMPARISON(κ) of ACCUMULATERESIDUE, and equals $2l_{\text{tot}} - 2$. At the peak, every repetition-code column of the chip is in use except the routing column and the unused exponent slot. Hence, at every d,

$$n_{\text{total}}(d) = \max_s \Gamma_{\text{physical}}(s) + \Gamma_{\text{global}}(d) + 2d + n_{\text{fac}} - l_{\text{tot}}, \tag{107}$$

the term $2d$ being the unused exponent slot and l_{tot} as in Equation (106).

We now check that this layout accommodates every slice. In ACCUMULATERESIDUE, the subroutine with the largest column count (Figure 51), the accumulation qubits sit in the compute zone with two free columns between consecutive qubits, and the residue

Parameter	Meaning	Value
n	Bit size of the RSA modulus N.	2048
s	Ekerå–Håstad parameter: a larger s shortens the exponent register but requires more runs.	8
n_e	Size of the exponent register.	1280
$\|P\|$	Number of primes in the RNS.	22 658
ℓ	Bit length of an RNS prime.	20
κ	Size of the accumulation register.	33
w_1	Window of the discrete-log computation.	6
w_3	Window of the exponentiation loops, $w_3 = w_{3a} = w_{3b}$ (Table 11).[a]	3
w_4	Window of the accumulation.	5

[a] ReduceExponent uses no window, hence there is no w_2.

Table 22: Algorithmic parameters of the RSA-2048 instance used in the resource estimates of Sections C.7 and C.8, taken from the RSA-2048 row of Table 5. The exponent register gathers the two Ekerå–Håstad registers, of $(1/2+1/(2s))n = 1152$ and $n/(2s) = 128$ qubits, hence $n_e = (1/2 + 1/s)n$ (Section A.5).

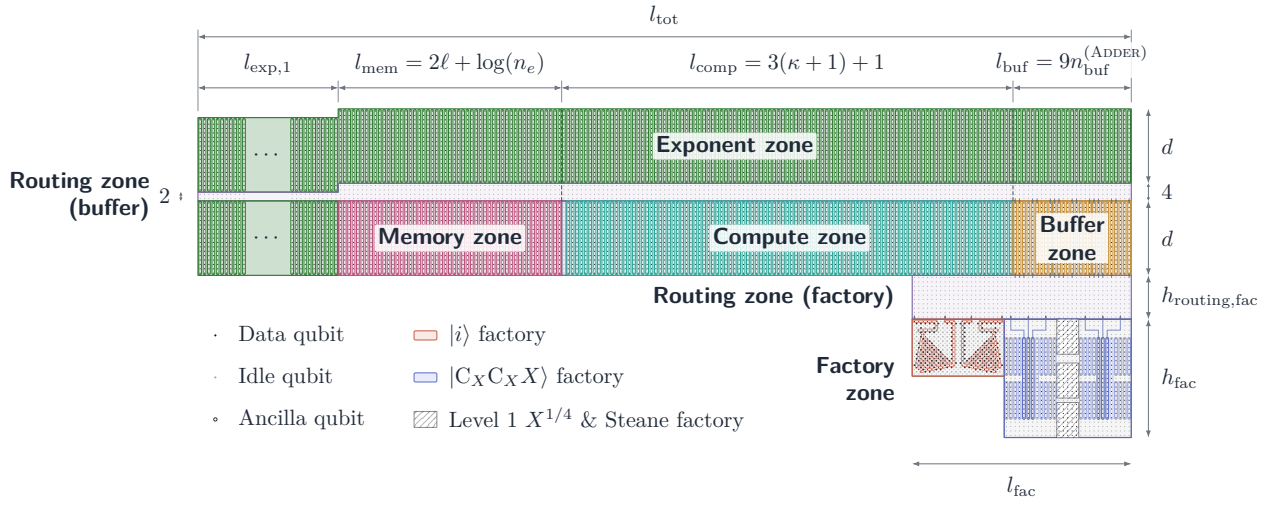


Figure 50: Global layout of the repetition-cat chip for RSA-2048. The compute zone holds the $3(\kappa+1)$ interleaved slots of the Adder plus one routing column on its left.

register, whose current w_4-bit window is the Lookup address, sits in the memory zone. The Lookup loads its k-th target immediately to the left of the k-th accumulation qubit, and its $(\kappa+1)$-th target three columns to the right of the κ-th one. The routing column and the free columns give these targets the Z_L access required by the Lookup, and the w_4-1 auxiliary qubits of its unary iteration fit in the buffer zone, of which the Lookup buffers use only 12 positions. Since every Adder writes the sum bit s_k in the column to the right of its input b_k and consumes b_k by a destructive Z-basis measurement, the Adder† and the Adder each shift the accumulation register by one column. The controls of the parity-Cnot gate $\mathrm{PC}X(\kappa)$ are prepared in $|+\rangle$ in the columns freed by these measurements, and the comparison bit of the UnderflowPhasefix is written into the last column of the compute zone, freed by the Z-basis measurement of the underflow qubit. The Phaseup auxiliary qubits fit in the buffer zone, and two transversal Cnot shifts restore the initial configuration. Since every Adder shifts the register it writes by one column, the same drift occurs in ComputeExponent, where S_i shifts by one column per iteration, and in the inner loops of ComputeResidue and ComputeResidue†, where the register written by the shifts by two columns per Adder†/Adder pair. It is compensated in the same way, by transversal Cnot shifts, except at the end of an inner loop, where the register moves anyway because the two ℓ-qubit registers swap roles. The other subroutines use the same operations. Outside the exponent zones, each slice must fit in the $l_{\text{mem}} + l_{\text{comp}} = 154$ columns. In ComputeResidue and ComputeResidue†, the widest operations, of size $\ell+1$, need $3(\ell+1)+1 = 64$ columns next to the $\kappa = 33$ packed accumulation qubits, i.e. $97 \le 103$ columns, while the memory zone holds $S_i \bmod (p_i - 1)$, the second ℓ-qubit register and the quotient register. The Q-

Subroutine	Operation	Repetition-code columns		
		Storage	Active	Total
ComputeExponent	Lookup(6,31)	1338	72	1410
	Q-Q Adder(31)	1313	120	1433
ReduceExponent	Q-C Adder(21)	1323	90	1413
	Q-Q Adder(20)	1324	87	1411
ComputeResidue	Lookup(6,20) (loading)	1338	54	1392
	Lookup(6,21)	1378	57	1435
	Q-Q Adder†(21)	1364	90	1454
	Q-Q Adder(20)	1365	87	1452
AccumulateResidue	Lookup(5,34)	1359	73	1432
	Q-Q Adder†(34)	1331	129	**1460**
	Q-Q Adder(33)	1332	126	1458
	Lookup(5,33)	1359	73	1432
	Q-Q Comparison(33)	1331	129	**1460**
	Phaseup(5)	1359	22	1381
ComputeResidue†	Lookup(6,21)	1378	57	1435
	Q-Q Adder†(21)	1364	90	1454
	Q-Q Adder(20)	1365	87	1452
	Lookup(6,20)	1378	54	1432
	Q-Q Comparison(20)	1364	90	1454
	Phaseup(6)	1378	26	1404
	Phaseup(6) (unload)	1344	26	1370
Last Phaseup	Phaseup(6)	1307	26	1333

Table 23: Breakdown by operations of the number of repetition-code columns for the factorization of an RSA-2048 number. Parameters are selected from Table 22. A repetition-code column is a vertical strip of the chip that holds one repetition code, counted with one ancilla per data qubit, and costs $2d$ physical qubits. All three counts (Storage, Active, Total) are in repetition-code columns. Storage counts the logical qubits held outside the operation, one repetition-code column each. Active counts the repetition-code columns of the current operation, including its buffers: $3m + 9n_{\text{buf}}^{(\text{Adder})}$ with $n_{\text{buf}}^{(\text{Adder})} = 3$ for an Adder of size m (Equation (88); a Comparison of size m is an Adder of size $m+1$), $n_{\text{col,Lookup}}^{(\text{rep})}(w, m, n_{\text{buf}}^{(\text{Lookup})}) = 2w + 3\lceil m/2 \rceil + 6n_{\text{buf}}^{(\text{Lookup})}$ with $n_{\text{buf}}^{(\text{Lookup})} = 2$ for a Lookup(w, m) (Equation (97)), including its own routing columns, and $n_{\text{col,Phaseup}}^{(\text{rep})}(w, n_{\text{buf}}^{(\text{Lookup})}) = 2^{\lceil w/2 \rceil} + 2^{\lfloor w/2 \rfloor} - 2 + 6n_{\text{buf}}^{(\text{Lookup})}$ with $n_{\text{buf}}^{(\text{Lookup})} = 2$ for a Phaseup(w) (Equation (99)). Total is the sum of Storage and Active. The routing rows and routing zones of the chip (Γ_{global}, Section C.7.1) and the factories ($n_{\text{fac}} = 1872$ physical qubits) are not included. The maximum, 1460 repetition-code columns, is reached by the Q-Q Adder†(34) and the Q-Q Comparison(33) of AccumulateResidue, and the global layout of Section C.7.1 is sized for this maximum.

Q Adder$(\ell + \log(n_e))$ of ComputeExponent needs 94 columns, so the accumulation register is parked in the memory zone during ReduceExponent†, ComputeExponent and ReduceExponent, next to the quotient register ($33 + 11 \leq 51$). The 27 positions of the buffer zone do not depend on the register size (Equation (92)) and serve every Adder, and the at most 5 Lookup and 8 Phaseup auxiliary qubits fit in the 15 positions left free by the 12 repetition codes of the Lookup buffers.

In the skew-tree QROM of [47], an And gate and the parity-Cnot gate that follows it act simultaneously on the same auxiliary qubit, the former through its X-parity measurements and Hadamard injection, the latter through the X-parity measurement that closes its parity. In the standalone layout of [47], they reach this qubit from opposite ends of its repetition code, the buffer lies below the register and a routing row runs above it. The global layout of Figure 50 has routing only above the compute zone, but the Lookup buffers and auxiliary qubits sit in the buffer zone, to the right of the compute zone. Both gates then fit in the four-row routing zone above it as follows. First, the $|0\rangle_L$ repetition code of the parity-Cnot gate is prepared in the lowest routing row, from which it reaches the Z-access columns of the Lookup targets. Second, the output of an And gate followed by a parity-Cnot gate is not moved to an auxiliary slot. Since the catalytic Hadamard injection is out-of-place, this output lies in one of the $|i\rangle$ columns of the buffer, where the parity-Cnot gate accesses it. It is measured in place by the measurement-based And uncomputation that immediately follows. Third, from left to right, the buffer zone holds the two Lookup buffers next to the compute zone, then the Lookup auxiliary qubits. Within each buffer, the

order is the $|i\rangle$ column that receives the AND output, the two other $|i\rangle$ columns, the $|\mathrm{C}_X\mathrm{C}_X X\rangle$ qubit consumed by the Hadamard injection, and the two other $|\mathrm{C}_X\mathrm{C}_X X\rangle$ qubits. The $|0\rangle_L$ code then ends at the leftmost column of the active buffer. The qubits accessed by the AND gate through the routing zone all lie outside its span: the address register, in the memory zone or in the top rectangle reached from above, the two $|\mathrm{C}_X\mathrm{C}_X X\rangle$ qubits measured against the controls, and the parent auxiliary qubit. The Hadamard injection itself only uses transversal CNOT gates between adjacent buffer columns. The two X-parity measurements of the AND gate are therefore routed in the three rows above the $|0\rangle_L$ code without crossing it, as in the three routing rows of [47, Fig. 14b]. The buffer not in use is refilled from below through the factory routing zone.

Between subroutines, at most two registers move (Figure 52): the accumulation register, between its interleaved position, the left end of the compute zone and the memory zone, and one ℓ-qubit register that switches between operand in the compute zone and address or storage in the memory zone. Within subroutines, the underflow qubits of REDUCEEXPONENT and REDUCEEXPONENT† join or leave the quotient register, and in COMPUTERESIDUE and COMPUTERESIDUE† the register written by the ADDERs moves to the memory zone to serve as a LOOKUP address. All moves use lattice surgery through the four-row routing zone, so they need no qubit beyond those of Equation (104). Their duration is bounded in the next paragraph.

iii Duration With $T_{\mathrm{surg}} = d$, $T_{\mathrm{cad}} = \max(d, 21)$ (Equation (83)), $T_{\mathrm{react}} = 10$, and $T_{\mathrm{CNOT}} = T_{\mathrm{meas}} = 1$ for a transversal CNOT gate and a transversal destructive measurement (Section C.1.1). Equation (93) then gives the duration of an n-bit ADDER,

$$T_{\mathrm{ADDER}}(n) = 2n\,T_{\mathrm{cad}}, \tag{108}$$

which also applies to the Q-C ADDER (neglecting the classical value preparation time) and to a COMPARISON of size n, implemented as an ADDER of size $n+1$. For $d \le 21$, this is $42n$. Equation (97) gives

$$T^{(\mathrm{rep})}_{\mathrm{LOOKUP}}(w, d) = (2^w - 1)\,d + (2^{w-1} - w)\,T_{\mathrm{cad}}, \tag{109}$$

that is $63d + 26T_{\mathrm{cad}}$ ($63d + 546$ for $d \le 21$) for the address size $w_1 = 2w_3 = 6$ and $31d + 11T_{\mathrm{cad}}$ ($31d + 231$) for $w_4 = 5$. PHASEUPs take $T^{(\mathrm{rep})}_{\mathrm{PHASEUP}}(w, d)$ cycles (Equation (99)), that is $7d + 8T_{\mathrm{cad}}$ for $w = 6$ and $3d + 5T_{\mathrm{cad}}$ for $w_4 = 5$. Summing over the numbers of calls N_{op} of Table 25, which depend only on Table 22, the duration of one shot is

$$\begin{aligned} T_{\mathrm{shot}}(d) &= \sum_{\mathrm{op}} N_{\mathrm{op}}\,T_{\mathrm{op}}(d) + T_{\mathrm{other}}(d) \\ &= 8.0909 \times 10^8\,T_{\mathrm{cad}} + 5.2251 \times 10^8\,d \\ &\quad + T_{\mathrm{other}}(d), \end{aligned} \tag{110}$$

where the term in T_{cad} is the cadence-limited part and $T_{\mathrm{other}}(d)$ collects the slices of Table 15 omitted from Table 25. In the color-changed convention (Section C.3), the $\mathrm{C}X(x)$, $M_X(x)$ and $P_{|0\rangle}(x)$ slices of Table 15 become parity-CNOT gates with x controls, $\mathrm{PC}X(x)$, Z-basis measurements $M_Z(x)$ and $|+\rangle$ preparations $P_{|+\rangle}(x)$. Their durations do not depend on the number of qubits involved. A $\mathrm{PC}X(x)$ is a single lattice surgery (Section C.2.1), $T_{\mathrm{PC}X} = T_{\mathrm{surg}} = d$, an $M_Z(x)$ is a transversal destructive Z-basis measurement, $T_{\mathrm{meas}} = 1$, and a $P_{|+\rangle}(x)$, a $|+\rangle_L$ preparation of x repetition codes in parallel, takes $T_{\mathrm{prep}} = T_{\mathrm{init}} = 1$ cycle (Section C.1.3). Summing the iteration counts of the corresponding rows of Table 15,

$$\begin{aligned} T_{\mathrm{other}}(d) &= N_{\mathrm{PC}X}\,d + N_{\mathrm{prep}}\,T_{\mathrm{init}} + N_{M_Z}\,T_{\mathrm{meas}} \\ &= N_{\mathrm{PC}X}\,d + N_{\mathrm{prep}} + N_{M_Z}, \end{aligned} \tag{111}$$

with

$$\begin{aligned} N_{\mathrm{PC}X} &= |P|\Big[4\log(n_e) + 6\big(\lceil \ell/w_3 \rceil - 2\big)\lceil \ell/w_3 \rceil \\ &\qquad + 2\lceil \ell/w_4 \rceil\Big], \end{aligned} \tag{112}$$

$$\begin{aligned} N_{M_Z} &= (|P| + 1)\lceil n_e/w_1 \rceil \\ &\quad + |P|\Big[5\big(\lceil \ell/w_3 \rceil - 2\big)\lceil \ell/w_3 \rceil \\ &\qquad + \lceil \ell/w_3 \rceil - 2 + \tfrac{3}{2}\lceil \ell/w_4 \rceil\Big], \end{aligned} \tag{113}$$

$$N_{\mathrm{prep}} = |P|\big(\lceil \ell/w_3 \rceil - 2\big). \tag{114}$$

For RSA-2048, $N_{\mathrm{PC}X} = 262|P|$, $N_{M_Z} = 400|P| + 214$ and $N_{\mathrm{prep}} = 5|P|$, so that $T_{\mathrm{other}}(d) \approx 5.94 \times 10^6 d + 9.18 \times 10^6$. These slices consume no factory output and are therefore not cadence-limited. Including them, $T_{\mathrm{shot}}(d) = 8.0909 \times 10^8\,T_{\mathrm{cad}} + 5.2844 \times 10^8\,d + 9.18 \times 10^6$, which reduces to $5.2844 \times 10^8\,d + 1.7000 \times 10^{10}$ for $d \le 21$.

The register moves of the Layout paragraph are not included in T_{shot}. We consider that the four routing rows above the memory and compute zones allow three qubits to move simultaneously, the crossings of their lattice surgeries being handled as in [46], so that a lattice-surgery move of an m-qubit register takes $\lceil m/3 \rceil d$ cycles. Between subroutines, the moves of Figure 52 take $65d$ cycles per prime, i.e. $65|P|d \approx 1.5 \times 10^6 d$ cycles, and at most $12|P|\lceil \kappa/3 \rceil d \approx 3.0 \times 10^6 d$ cycles for any order of the moves. Within subroutines, the $2|P|\log(n_e) \approx 5.0 \times 10^5$ single-qubit moves of the underflow qubits and the at most $3|P|(\lceil \ell/w_3 \rceil - 2) \approx 3.4 \times 10^5$ moves of ℓ-qubit registers in COMPUTERESIDUE and COMPUTERESIDUE† take at most $\approx 2.9 \times 10^6 d$ cycles. The transversal CNOT shifts that compensate the drift of the ADDERs, about 9×10^6 single-column register shifts per shot, take a number of cycles independent of d, at most about 3×10^7 cycles, i.e. less than 0.15% of T_{shot}. With the schedule of Figure 52, the moves together stay below 0.5% of T_{shot} at every d (below 0.6% with the bound

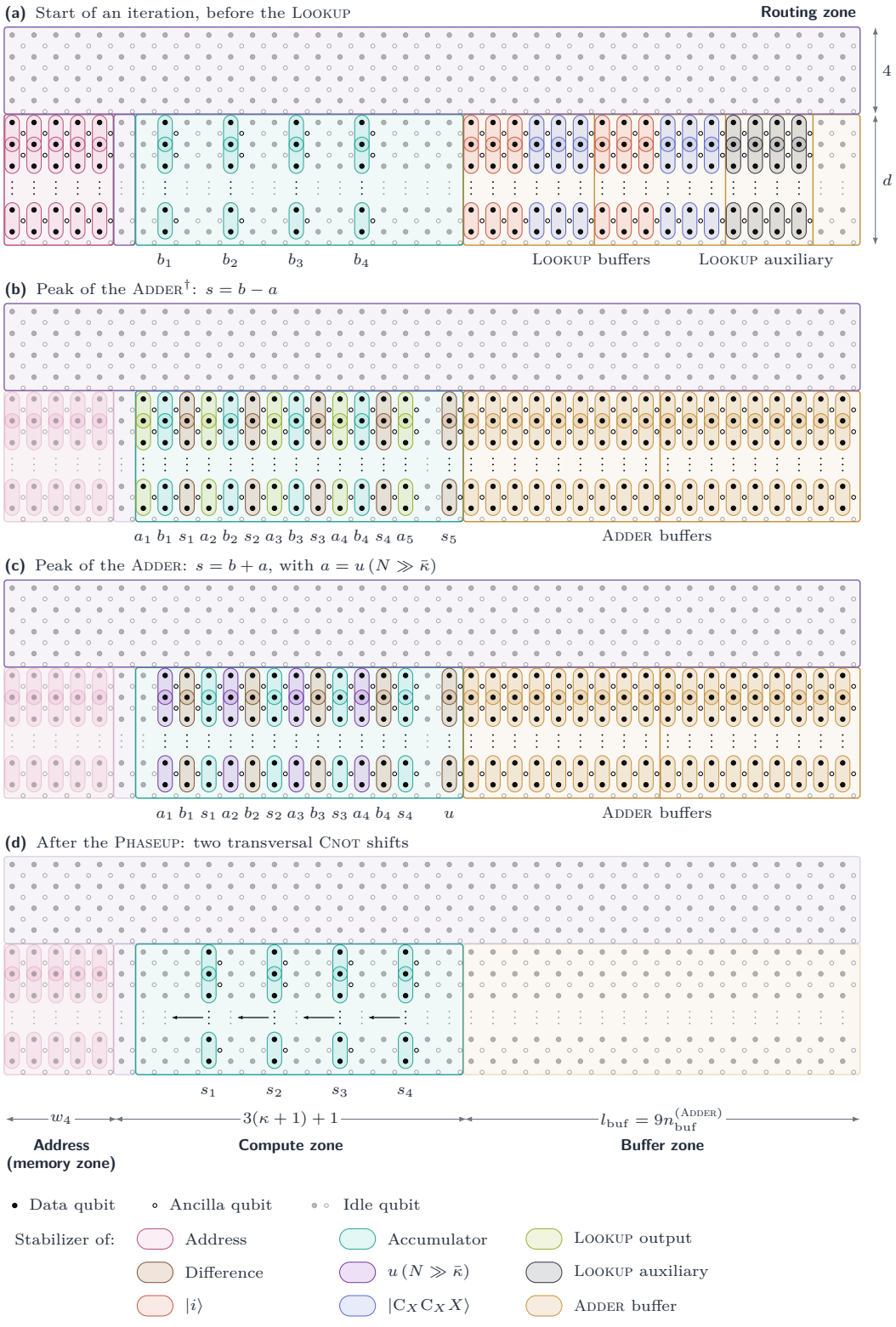


Figure 51: Repetition-cat layouts at four time steps of ACCUMULATERESIDUE (Figure 32), drawn for $\kappa = 4$ and $n_{\text{buf}}^{(\text{ADDER})} = 2$. Columns a_k, b_k and s_k are the slots of the ADDER† in (a) and (b), and of the ADDER, one column to the right, in (c) and (d). Zones that a step does not use are faded. In (a), $b_{\kappa+1} = 0$ is not allocated (Equation (86)). In (b), between the last injection and the first correction (Figure 49), the top output bit $s_{\kappa+1}$ is the underflow qubit u. In (c), after the output of the LOOKUP is measured, $\mathrm{PC}X(\kappa)$ loads $a = u\,(N \gg \bar{\kappa})$ into the columns freed by the b_k of (b). In (d), the shifts move the accumulator two columns left, restoring (a).

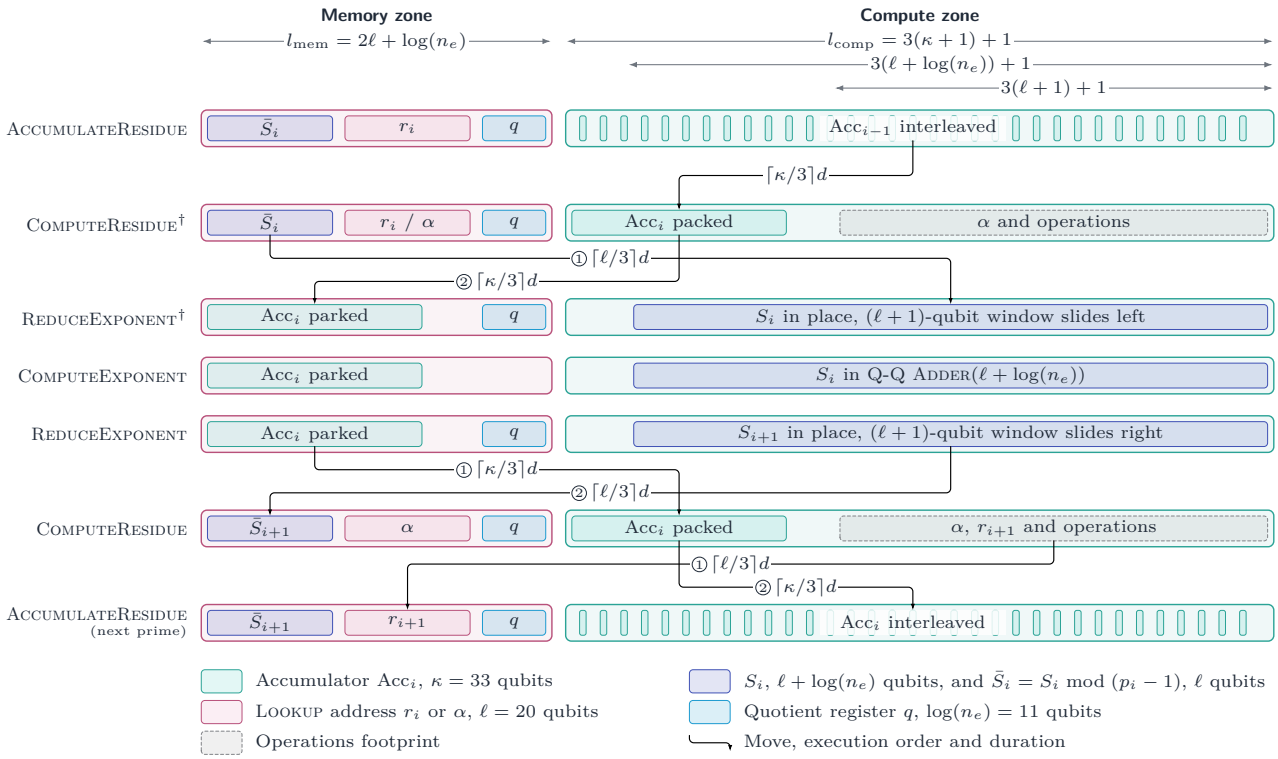


Figure 52: Register movements through the six subroutines of one prime on the repetition-cat chip of Figure 50 (RSA-2048), drawn to scale with $l_{\text{mem}} = 51$ memory and $l_{\text{comp}} = 103$ compute columns. Rows display the layout during each subroutine, starting from AccumulateResidue, with their input register values. Arrows represent the moves of each transition, their execution order, and the outlined boxes the associated qubit footprints. Moves within a subroutine, of the ℓ-qubit registers in ComputeResidue and ComputeResidue† and of the quotient qubits in ReduceExponent and ReduceExponent†, are not drawn. Similarly, the exponent and buffer zones are not drawn, since there are no register movements associated. The accumulator is stored in the memory zone during ReduceExponent†, ComputeExponent and ReduceExponent, where S_i occupies $3(\ell + \log(n_e)) + 1 = 94$ of the $l_{\text{comp}} = 103$ compute columns. Each lattice surgery move of an m-qubit register requires $\lceil m/3 \rceil d$ cycles, such that the moves between the subroutines of a prime have a total duration of $4\lceil \kappa/3 \rceil d + 3\lceil \ell/3 \rceil d = 65d$ cycles.

for any order of the moves), and we neglect them. If only two qubits or one qubit can move simultaneously in some cases, these moves take longer, but the impact on the duration is small and we do not consider it. The final inverse quantum Fourier transform is not included in T_{shot} either. In its semi-classical form (Section A.3), each of the n_e exponent qubits receives one rotation, whose angle depends on the outcomes of the qubits already measured, and is then measured by a transversal destructive Z-basis measurement. Each rotation is applied by phase kickback, the exponent qubit controls the addition of a classically known n_φ-bit constant into a phase-gradient register of n_φ qubits [104], prepared once before the transform. A rotation therefore costs one Q-Q Adder(n_φ), i.e. $n_\varphi - 1$ Toffoli gates, plus the $|+\rangle$ preparation and the parity-Cnot gate that load the constant, one measurement and one reaction time. The transform thus takes $n_e\,[T_{\text{Adder}}(n_\varphi) + T_{\text{init}} + d + T_{\text{meas}} + T_{\text{react}}]$ cycles (Equation (108)). For $n_\varphi = 25$, this is 1.38×10^6 cycles at $d = 15$, i.e. 5×10^{-5} of T_{shot}, and 3.1×10^4 Toffoli gates, i.e. less than 5×10^{-5} of the Toffoli count per shot, and we neglect both, as well as the one-time preparation of the phase-gradient register.

iv Volume We combine the column counts of the Layout paragraph and the durations of the Duration paragraph into the probability that a shot suffers a logical error. One physical qubit-cycle corresponds to one physical qubit being allocated during one error-correction cycle. The physical volume of a shot is

$$\text{Vol}_{\text{physical}} = \sum_s \big[2d\, n_{\text{col}}(s) - n_{e,2} + \Gamma_{\text{global}}\big] \times T_{\text{error correction}}(s), \tag{115}$$

with $n_{\text{col}}(s)$ the number of repetition-code columns of slice s (Table 23), $n_{e,2} = n_e - l_{\text{exp},1}$ (730 for RSA-2048) the number of exponent qubits in the top rectangle, whose ancillae above them are not needed since nothing lies above them (the unused exponent slot is not in $n_{\text{col}}(s)$, so only these 730 of the $l_{\text{tot}} = 731$ exposed ancillae are removed here), Γ_{global} the global routing of Equation (103), and $T_{\text{error correction}}(s)$ the duration of the slice in error-correction cycles. The repetition-code columns of a slice are those of the

logical qubits stored outside the current operation and those of the operation itself, including its buffers, its unallocated slots and, for a LOOKUP, the routing columns included in $n^{(\mathrm{rep})}_{\mathrm{col,LOOKUP}}$ (Equation (97)). The factory routing zone is counted even though the factories are not, since it carries the resource states after they leave the factories, so its errors are not covered by the factory output infidelities. Equation (115) overestimates the volume, since routing qubits are not all allocated in every cycle, nor are the unallocated slots of an operation. The physical volume is translated into an effective logical volume by dividing it by $2d-1$, the size of the repetition code for which p_{logical} is defined [16] (Section C.1.3), and by d, to translate error-correction cycles into logical cycles:

$$\begin{aligned}\mathrm{Vol}^{(\mathrm{eff})}_{\mathrm{logical}} &= \frac{\mathrm{Vol}_{\mathrm{physical}}}{d(2d-1)} \\ &= \frac{1}{d}\sum_s \frac{2d\,n_{\mathrm{col}}(s) - n_{e,2} + \Gamma_{\mathrm{global}}}{2d-1} \\ &\qquad \times T_{\mathrm{error\,correction}}(s).\end{aligned} \tag{116}$$

In the effective logical volume, global routing therefore counts as $\Gamma_{\mathrm{global}}/(2d-1)$ additional effective logical qubits throughout the shot, whereas the magic-state factories are not counted, since their logical error rate is accounted for separately. We denote by $\epsilon_{\mathrm{logical}}$ the probability that at least one logical error occurs during one shot,

$$\begin{aligned}\epsilon_{\mathrm{logical}} &= 1 - (1-p_{\mathrm{logical}})^{\mathrm{Vol}^{(\mathrm{eff})}_{\mathrm{logical}}} \\ &\simeq p_{\mathrm{logical}}\,\mathrm{Vol}^{(\mathrm{eff})}_{\mathrm{logical}},\end{aligned} \tag{117}$$

where p_{logical} is the logical error rate per logical cycle of one logical qubit. We also introduce T_{logical}, the duration of the computation in logical cycles, and $n^{(\mathrm{eff})}_{\mathrm{logical}}$, the effective number of logical qubits,

$$\begin{aligned}T_{\mathrm{logical}} &= \frac{T_{\mathrm{error\,correction}}}{d}, \\ n^{(\mathrm{eff})}_{\mathrm{logical}} &= \frac{\mathrm{Vol}^{(\mathrm{eff})}_{\mathrm{logical}}}{T_{\mathrm{logical}}} = \frac{2d\,\bar{n}_{\mathrm{col}} - n_{e,2} + \Gamma_{\mathrm{global}}}{2d-1},\end{aligned} \tag{118}$$

where $T_{\mathrm{error\,correction}} = \sum_s T_{\mathrm{error\,correction}}(s)$ and $\bar{n}_{\mathrm{col}}$ is the time average of $n_{\mathrm{col}}(s)$ over the shot.

For RSA-2048, each operation row s of Tables 23 and 25 is repeated N_s times and lasts $T_s(d)$ cycles (Equations (99), (108) and (109)). Since $n_{\mathrm{col}}(s)$ does not depend on d, Equation (115) reads

$$\mathrm{Vol}_{\mathrm{physical}}(d) = 2d\,X(d) + \left[\Gamma_{\mathrm{global}}(d) - n_{e,2}\right] T_{\mathrm{shot}}(d), \tag{119}$$

with

$$\begin{aligned}X(d) &= \sum_s N_s\,n_{\mathrm{col}}(s)\,T_s(d) + n_{\mathrm{col,other}}\,T_{\mathrm{other}}(d) \\ &= 1.1604\times10^{12}\,T_{\mathrm{cad}} + 7.5024\times10^{11}\,d \\ &\quad + 1.3398\times10^{10},\end{aligned} \tag{120}$$

$\Gamma_{\mathrm{global}}(d)$ given by Equation (105) and $T_{\mathrm{shot}}(d)$ by Equation (110). Keeping the factory routing term $2h_{\mathrm{routing,fac}}\,l_{\mathrm{fac}}$ symbolic, since its value is set in Section C.7.2, this gives

$$\begin{aligned}\mathrm{Vol}_{\mathrm{physical}}(d) &= 1.502\times10^{12}\,d^2 + 2.322\times10^{12}\,d\,T_{\mathrm{cad}} \\ &\quad + 1.569\times10^{12}\,d + 2.361\times10^{12}\,T_{\mathrm{cad}} \\ &\quad + 2.678\times10^{10} \\ &\quad + 2h_{\mathrm{routing,fac}}\,l_{\mathrm{fac}}\,T_{\mathrm{shot}}(d).\end{aligned} \tag{121}$$

These coefficients rest on three assumptions. First, the slices of $T_{\mathrm{other}}(d)$ are assigned the peak column count of the computation, $n_{\mathrm{col,other}} = 1460$, an upper bound since these slices are always smaller than the adjacent ADDER or LOOKUP operations. Second, REDUCEEXPONENT†, which applies the operations of REDUCEEXPONENT in reverse, has the same column counts. Third, the active columns of a LOOKUP are its $n^{(\mathrm{rep})}_{\mathrm{col,LOOKUP}}$ columns (Equation (97)). When a LOOKUP loads an ADDER input, its targets sit in the interleaved slots of the latter (Figure 51) and each of them uses its own free neighboring column as routing, so the LOOKUP uses up to 16 more repetition-code columns than counted. These slices stay below the peak of 1460 columns, and the volume is underestimated by about 0.4%.

C.7.2 Evaluation at $p_Z = 10^{-3}$ and $p_Z = 5\times10^{-4}$

We now evaluate the estimate for RSA-2048 at the distances $d = 13$, 15 and 17, and at the phase-flip probabilities $p_Z = 10^{-3}$ and $p_Z = 5\times10^{-4}$. Our main result is at $p_Z = 5\times10^{-4}$ and $d = 15$. As in Section C.7.1, operations are written in the color-changed convention of Section C.3.

i Physical qubit count, duration and volume

We use the factories of Table 19. Their layout and production time do not depend on p_Z, only the infidelity of their output states does (Section C.7.2.ii). For all distances considered here, $T_{\mathrm{cad}} = \max(d, 21) = 21$ is set by the $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory, and Equation (82) gives one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory and two $|i\rangle$ factories. The factory zone has length $l_{\mathrm{fac}} = w^{|\mathrm{C}_X\mathrm{C}_X X\rangle}_{\mathrm{fac}} + 2w^{|i\rangle}_{\mathrm{fac}} = 29 + 2\times10.5 = 50$ and contains $n_{\mathrm{fac}} = 1566 + 2\times153 = 1872$ physical qubits, as in Section C.5.7. The factory routing zone has height $h_{\mathrm{routing,fac}} = 10$, hence $2h_{\mathrm{routing,fac}}\,l_{\mathrm{fac}} = 1000$ in Equations (105) and (106).

With these values, Equations (106), (110), (116) and (121) give the physical qubit count, the duration of one shot and the effective logical volume listed in Table 24 for $d = 13$, 15 and 17. At $d = 15$ and at the peak of Table 23, the 1460 repetition-code columns hold $1460\times2d = 43\,800$ physical qubits, of which $n_e\times2d = 38\,400$ store the exponent. The global routing $\Gamma_{\mathrm{global}} = 4678$ of Equation (105) splits into 1000 (factory routing zone), 2200 (routing zone between the two exponent zones), 1448 (routing zone

above the memory, compute and buffer zones) and 30 (routing column of the compute zone). Adding the factories ($n_{\text{fac}} = 1872$), the global routing and the unused exponent slot ($2d = 30$), and subtracting the $l_{\text{tot}} = 731$ exposed ancillae gives the 49 649 physical qubits of Equation (106), in agreement with Equation (107). One shot takes $T_{\text{shot}} = 2.493 \times 10^{10}$ cycles, about 6.92 hours (not accounting for classical precomputation).

At $d = 15$, $\Gamma_{\text{global}} = 4678$ amounts to 161 effective logical qubits, $T_{\text{logical}} = 1.66 \times 10^9$ and $n^{(\text{eff})}_{\text{logical}} = 1615$, of which $\bar{n}_{\text{col}} = 1430$. For $d < 21$, the cadence sets about two thirds of the shot length, so that the number of logical cycles T_{shot}/d grows as d decreases, and $\text{Vol}^{(\text{eff})}_{\text{logical}}$ grows somewhat more slowly than $1/d$. Going from $d = 17$ to $d = 13$ thus saves 21% of the physical qubits but shortens a shot by only 8%, while the effective logical volume grows by 24%. The logical error probability per shot grows much faster, since p_{logical} increases by a factor of about $1/(Bp_Z)^C$ per step of two in d (Equation (122)). From $d = 17$ to $d = 13$, $\epsilon_{\text{logical}}$ rises from 2.6×10^{-5} to 0.157 at $p_Z = 5 \times 10^{-4}$, and from 0.0125 to almost 1 at $p_Z = 10^{-3}$ (Table 24). A smaller distance therefore trades space for retries only within a narrow range of d (Section C.7.2.ii).

ii Logical error budget, number of shots and total runtime We take the logical error model of the repetition code under generic circuit-level noise from [16, Supplementary Table 1(b)], in which every operation of the syndrome-extraction circuit has phase-flip probability p_Z. The probability that a logical error occurs on one logical qubit during one logical cycle, i.e. d error-correction cycles, is

$$p_{\text{logical}}(p_Z, d) = A\, d\, (B\, p_Z)^{C\lfloor (d+1)/2 \rfloor}, \qquad (122)$$

with $A = 0.12$, $B = 23$ and $C = 0.99$. This fit is obtained from simulations at logical error probabilities above about 10^{-8} [16, Supplementary Fig. 2(b)]. The values of Table 24 extrapolate it to the regime relevant here. By Equation (117), the probability per shot that a logical error occurs on the data and routing qubits is $\epsilon_{\text{logical}} = 1 - (1 - p_{\text{logical}}(p_Z, d))^{\text{Vol}^{(\text{eff})}_{\text{logical}}}$, with $p_{\text{logical}}(p_Z, d)$ given by Equation (122). We use this exact form, also given in Table 24, since $\epsilon_{\text{logical}}$ can exceed a few percent at small d, where the linear approximation of Equation (117) no longer holds.

The output infidelities of the factories of Table 19 are 1.48×10^{-11} ($p_Z = 10^{-3}$) and 2.02×10^{-13} ($p_Z = 5 \times 10^{-4}$) for the $|\text{C}_X\text{C}_X X\rangle$ factory, and 3.1×10^{-12} and 9.43×10^{-13} for the $|i\rangle$ factory, a color code of distance $d_i = 7$. Counting the Toffoli gates of each operation as in Sections C.5.6 and C.6, with a Comparison of size n as an Adder of size $n + 1$, and Adder† and Q-C Adders as Adders of the same size, the calls of Table 25 give 7.56×10^8 Toffoli gates per shot: 2.81×10^8 in Adders and Comparisons (37%), 4.62×10^8 in Lookups (61%) and 1.33×10^7 in Phaseups (1.8%). The $|\text{C}_X\text{C}_X X\rangle$ factories therefore cause a failure during a shot with probability $\epsilon_{|\text{C}_X\text{C}_X X\rangle} \approx 1.1\%$ at $p_Z = 10^{-3}$ and 1.5×10^{-4} at $p_Z = 5 \times 10^{-4}$. With the $|i\rangle$-state counts of the same sections, catalyst seeds included, the calls give 2.17×10^9 $|i\rangle$ states per shot: 1.16×10^9 in Adders and Comparisons (54%), 9.40×10^8 in Lookups (43%) and 6.64×10^7 in Phaseups (3.1%), of which 5.46×10^7 are catalyst seeds. The $|i\rangle$ factories therefore cause a failure during a shot with probability $\epsilon_{|i\rangle} \approx 0.67\%$ at $p_Z = 10^{-3}$ and 0.20% at $p_Z = 5 \times 10^{-4}$. These two contributions do not depend on d. The per-shot failure probability is $\epsilon = 1 - (1 - \epsilon_{\text{logical}})(1 - \epsilon_{|\text{C}_X\text{C}_X X\rangle})(1 - \epsilon_{|i\rangle})$.

We select the distance with the following criterion: among the distances for which the per-shot failure probability ϵ is below 10%, we choose the one with the fewest physical qubits. On the repetition-cat architecture, n_{total} grows with d (Equation (106)), so this is the smallest odd distance with $\epsilon < 10\%$. A shot with a logical error is caught and retried (Section A.9.0.vii), so ϵ multiplies the expected number of shots by $1/(1 - \epsilon)$, and the threshold limits this increase to 11%. At $p_Z = 5 \times 10^{-4}$, $d = 13$ gives $\epsilon \approx 16\%$, so we choose $d = 15$, which gives $\epsilon_{\text{logical}} = 0.211\%$ and $\epsilon \approx 0.43\%$. At this point, the data and routing qubits (0.21%) and the $|i\rangle$ factories (0.20%) contribute almost equally to the per-shot failure, while the $|\text{C}_X\text{C}_X X\rangle$ factories (0.015%) are negligible. At $p_Z = 10^{-3}$, $d = 15$ gives $\epsilon \approx 41\%$, so the same criterion selects $d = 17$, with $\epsilon \approx 3.0\%$. The selected distances are therefore the same for any threshold between 3.0% and 16%. For $s = 8$, the average number of shots required is 9.2, as explained in Section A.9. Accounting for the per-shot failure probability ϵ, it becomes $9.2/(1 - \epsilon) = 9.24$ at $p_Z = 5 \times 10^{-4}$ and $d = 15$, which brings the total duration to 9.24×6.92 h ≈ 64.0 h ≈ 2.67 days. At $p_Z = 5 \times 10^{-4}$, $d = 13$ would save 12% of the physical qubits (43 801 versus 49 649) at the cost of 13% more runtime (3.02 versus 2.67 days). The shots are independent, so they can run in parallel on several quantum computers, which reduces the total runtime accordingly.

iii Duration breakdown Table 25 gives the number of cycles of each operation at $d = 15$. ComputeExponent represents 54% of the total duration. Subroutine shares below include the $|+\rangle$ preparation, Z-basis measurement and parity-Cnot slices omitted from Table 25, and therefore slightly exceed the sums of its rows. ComputeResidue† is about 3 times longer than ComputeResidue (30.0% versus 10.5%). This is expected since ComputeResidue† must properly uncompute the modular exponentiation, which requires two inner loops, each including a phase correction. ComputeResidue, however, needs

d	n_{total}	T_{shot}		$\text{Vol}^{(\text{eff})}_{\text{logical}}$	$p_Z = 10^{-3}$		$p_Z = 5 \times 10^{-4}$	
		error-correction cycles	hours		p_{logical}	$\epsilon_{\text{logical}}$	p_{logical}	$\epsilon_{\text{logical}}$
13	43 801	2.387×10^{10}	6.63	3.02×10^{12}	6.92×10^{-12}	$> 99.5\%$	5.67×10^{-14}	15.7%
15	49 649	2.493×10^{10}	6.92	2.68×10^{12}	1.91×10^{-13}	40.0%	$\mathbf{7.87 \times 10^{-16}}$	**0.211**%
17	55 497	2.598×10^{10}	7.22	2.43×10^{12}	$\mathbf{5.16 \times 10^{-15}}$	**1.25**%	1.07×10^{-17}	2.6×10^{-3}%

Table 24: Physical resources and logical error budget of the repetition-cat architecture for RSA-2048, at $p_Z = 10^{-3}$ and $p_Z = 5 \times 10^{-4}$. n_{total} is given by Equation (106), T_{shot} by Equation (110) and $\text{Vol}^{(\text{eff})}_{\text{logical}}$ by Equations (116) and (121), with the factories of Table 19. p_{logical} is the logical error probability per logical cycle of Equation (122), and $\epsilon_{\text{logical}}$ the probability of a logical error on the data and routing qubits during one shot (exact form of Equation (117)). The entry $> 99.5\%$ denotes a probability above 0.995. The main result is at $p_Z = 5 \times 10^{-4}$ and $d = 15$.

a single inner loop without any phase correction. Finally, ReduceExponent (including ReduceExponent†) and AccumulateResidue represent 3.5% and 1.7% of the total duration respectively. The time spent on addition-like operations and on Lookups is comparable, accounting respectively for 49.3% and 48.5% of the total duration. Other operations have a much smaller contribution, with Phaseups representing 1.8% of the duration and the PCX, M_Z and $P_{|+\rangle}$ slices 0.39% combined. The distribution depends only weakly on d. The share of the Lookups goes from 46% at $d = 13$ to 50% at $d = 17$, since a Lookup has a larger surgery-limited part than an Adder, whose duration is entirely cadence-limited. We do not account for the reaction time between the M_Z of underflow qubits and the conditional application of an UnderflowPhasefix, since its contribution to the total duration is negligible.

C.8 LDPC-cat architecture

This section estimates the physical resources of the computation for RSA-2048 on the LDPC-cat architecture, following the method of Section C.7. Section C.8.1 describes the layout, the duration and the volume of one shot at arbitrary distances d_{rep} and d_{LDPC} of the repetition and LDPC layers, and Section C.8.2 evaluates the estimate at $p_Z = 5 \times 10^{-4}$ and $p_Z = 10^{-3}$, where the selected distances are $(d_{\text{rep}}, d_{\text{LDPC}}) = (13, 16)$ and $(17, 22)$, and compares it with the repetition-cat architecture. As in Section C.7, operations are written in the color-changed convention of Section C.3.

The LDPC-cat architecture differs from the repetition-cat architecture in two ways. First, the logical data qubits are stored in a single LDPC-cat code block, in which the operator Z_L of a logical qubit is only accessible through transversal Cnot gates onto a repetition code of the repetition layer that covers its support, which must then undergo $d^\star = \max(d_{\text{rep}}, d_{\text{LDPC}})$ rounds of stabilizer measurements before it is read out. Z-basis measurements and register moves, which take one cycle or are neglected on the repetition-cat architecture (Section C.7.1), therefore take $T^{(\text{LDPC})}_{\text{surg}}$ cycles each and are no longer negligible (Section C.8.1.iii). Second, the chip has two layers, with two distances, two logical error models and two volumes (Section C.8.1.v).

C.8.1 Layout, duration and volume at distances d_{rep} and d_{LDPC}

i Introduction We divide the algorithm into the temporal slices of Table 15, with the algorithmic parameters of Table 22 and the factories of Section C.4, and assume, as in Section C.7.1, an error-correction cycle of 1 μs, a reaction time $T_{\text{react}} = 10$ cycles and no bit flip during the computation.

Two distances enter the estimate. The LDPC distance d_{LDPC} is chosen among the codes of Table 20, which fixes the height H of the block and the number Δ_{LDPC} of qubits of its staircases. The repetition distance d_{rep} applies to every repetition code of the repetition layer: the buffers, the factory outputs, the routing codes and the auxiliary qubits of Lookups and Phaseups.

The number of rounds of a lattice surgery sets its timelike distance, which protects the outcome of the surgery against chains of measurement errors. Every lattice surgery involving the LDPC layer, such as the Z-basis measurements and register moves of Section C.8.1.iii, the surgeries of the Adder, the And gates and parity-Cnot gates of Lookups, and the And gates and multi-target CX gates of Phaseups, lasts $T^{(\text{LDPC})}_{\text{surg}}$ cycles (Equation (78)). A surgery confined to the repetition layer, such as the loading of factory outputs into the buffers, lasts d_{rep} cycles.

The steps that consume factory outputs are surgeries involving the LDPC layer, and they start at the cadence of Equation (83) with $T_{\text{surg}} = T^{(\text{LDPC})}_{\text{surg}}$,

$$\begin{aligned} T_{\text{cad}} &= \max\left(T^{(\text{LDPC})}_{\text{surg}}, T^{|\text{C}_X\text{C}_X X\rangle}_{\text{prod}}, T^{|i\rangle}_{\text{prod}}\right) \\ &= \max(T^{(\text{LDPC})}_{\text{surg}}, 21), \end{aligned} \tag{123}$$

so that, as in Section C.7.1, the $|\text{C}_X\text{C}_X X\rangle$ factory sets the cadence $T_{\text{cad}} = 21$ for $T^{(\text{LDPC})}_{\text{surg}} \leq 21$, that is for every pair of distances considered here with $d_{\text{LDPC}} \leq 16$, whereas the surgeries set it for $d_{\text{LDPC}} = 22$, where $T^{(\text{LDPC})}_{\text{surg}} = T_{\text{cad}} = 22$. In both cases, Equation (82) gives one $|\text{C}_X\text{C}_X X\rangle$ factory and two $|i\rangle$ factories. Each buffer reservation contains one loading surgery,

Subroutine	Operation	Calls	Cycles/call	Cycles	Share
ComputeExponent	Lookup(6, 31)	4849026	1491	7.23×10^9	29.00%
	Q-Q Adder(31)	4849026	1302	6.31×10^9	25.33%
ReduceExponent	Q-C Adder(21)	249238	882	2.20×10^8	0.88%
	Q-Q Adder(20)	249238	840	2.09×10^8	0.84%
ComputeResidue	Lookup(6, 20) (loading)	22658	1491	3.38×10^7	0.14%
	Lookup(6, 21)	793030	1491	1.18×10^9	4.74%
	Q-Q Adder†(21)	793030	882	6.99×10^8	2.81%
	Q-Q Adder(20)	793030	840	6.66×10^8	2.67%
AccumulateResidue	Lookup(5, 34)	90632	696	6.31×10^7	0.25%
	Q-Q Adder†(34)	90632	1428	1.29×10^8	0.52%
	Q-Q Adder(33)	90632	1386	1.26×10^8	0.50%
	Lookup(5, 33)	45316	696	3.15×10^7	0.13%
	Q-Q Comparison(33)	45316	1428	6.47×10^7	0.26%
	Phaseup(5)	90632	150	1.36×10^7	0.05%
ComputeResidue†	Lookup(6, 21)	1586060	1491	2.36×10^9	9.49%
	Q-Q Adder†(21)	1586060	882	1.40×10^9	5.61%
	Q-Q Adder(20)	1586060	840	1.33×10^9	5.34%
	Lookup(6, 20)	793030	1491	1.18×10^9	4.74%
	Q-Q Comparison(20)	793030	882	6.99×10^8	2.81%
	Phaseup(6)	1586060	273	4.33×10^8	1.74%
	Phaseup(6) (unload)	22658	273	6.19×10^6	0.02%
ReduceExponent†	Q-C Adder(21)	249238	882	2.20×10^8	0.88%
	Q-Q Adder(20)	249238	840	2.09×10^8	0.84%
Last Phaseup	Phaseup(6)	214	273	5.84×10^4	0.00%

Table 25: Breakdown of error-correction cycle counts by operation for the factorization of an RSA-2048 number. All logical data qubits are encoded in a repetition code of distance 15. Time slices involving a $|+\rangle$ preparation, a Z-basis measurement or a parity-Cnot gate are not reported in this table for clarity, as they have a small contribution of 0.39% to the total duration. Their contribution is still accounted for in the total duration and in the total volume. *Cycles/call* refers to the number of error-correction cycles required for one call of an operation, while *Cycles* gives the total number of error-correction cycles required for all calls of this operation.

confined to the repetition layer and lasting $T_{\text{surg}} = d_{\text{rep}}$, and one injection or correction surgery, which involves the LDPC layer and lasts $T_{\text{surg}}^{(\text{LDPC})}$. Replacing one of the two T_{surg} terms of Equation (92) accordingly, with $\max\big(T_{\text{prep}}^{|\text{AA}\rangle}, T_{\text{prep}}^{|\text{DC}_X X\rangle}\big) = 9$ (Section C.5.4) and $T_{\text{react}} + T_{\text{meas}} = 11$, the reservation time becomes

$$\begin{aligned} T_{\text{buf}} &= d_{\text{rep}} + T_{\text{surg}}^{(\text{LDPC})} + \max\big(T_{\text{prep}}^{|\text{AA}\rangle}, T_{\text{prep}}^{|\text{DC}_X X\rangle}\big) \\ &\quad + T_{\text{react}} + T_{\text{meas}} \\ &= d_{\text{rep}} + T_{\text{surg}}^{(\text{LDPC})} + 20, \end{aligned} \tag{124}$$

and the number of Adder buffers is $n_{\text{buf}}^{(\text{Adder})} = \lceil T_{\text{buf}}/T_{\text{cad}} \rceil$. For every pair of distances considered in Table 28, $43 \leq T_{\text{buf}} \leq 61$ and $n_{\text{buf}}^{(\text{Adder})} = 3$, as on the repetition-cat architecture. Lookups and Phaseups use $n_{\text{buf}}^{(\text{Lookup})} = 2$ resource-state buffers of six repetition codes each (Section C.6).

Physical qubits are counted as in Section C.7.1, with one difference. In the LDPC layer, each physical qubit lies in the support of the logical operators of several logical qubits, so that its qubits cannot be attributed to individual logical qubits. We therefore count the whole LDPC layer as allocated during the whole shot. On the repetition layer, as in Section C.7.1, a slice s costs $2d_{\text{rep}}\, n_{\text{col}}(s)$ physical qubits, with $n_{\text{col}}(s)$ the number of repetition-code columns allocated by its operation, i.e. its buffers and auxiliary qubits (Buffer zone column of Table 26), since its data qubits are in the LDPC layer. All other qubits of the repetition layer, except the factories, are never counted within an operation, including the access zone and the routing zones that an operation uses: they form the global routing Γ_{global}, present during the whole shot, and the zone Γ_{QFT}, used only by the final inverse quantum Fourier transform.

ii Layout The global layout, shown in Figure 53, is determined by the requirements of the AccumulateResidue subroutine. The chip is a flip chip of two facing layers (Section C.1.3). The LDPC layer holds one open-boundary LDPC-cat code block [16] of H rows and L columns, which stores all logical data qubits. Its upper rows extend laterally into the left and right staircases of Section C.1.3, of Δ_{LDPC} qubits in total (Table 20). The operator X_L of a log-

ical qubit is the weight-one operator of a tip qubit. The tips form the bottom two rows of the block, with two logical qubits per column. The operator Z_L is supported on a set that spreads upward over at most $2H-1$ columns. The block is divided into an exponent zone, a memory zone and a compute zone of

$$l_{\rm exp} = \frac{n_e}{2}, \quad l_{\rm mem} = \left\lceil \frac{2\ell + \log(n_e)}{2} \right\rceil, \\ l_{\rm comp} = \left\lceil \frac{3(\kappa+1)}{2} \right\rceil \tag{125}$$

columns, so that $L = l_{\rm exp} + l_{\rm work}$, with $l_{\rm work} = l_{\rm mem} + l_{\rm comp}$ the length of the *work zone*, made of the memory and compute zones. The work zone holds the same logical qubits as on the repetition-cat architecture. The compute zone keeps the $3(\kappa+1)$ interleaved slots of the ADDER† of ACCUMULATERESIDUE, but needs no routing column, since Z_L access comes from the repetition layer. The $[n_L, k_L, d_{\rm LDPC}]$ code has $n_L = HL + \Delta_{\rm LDPC}$ data qubits, one ancilla per check and $k_L = 2L$ logical qubits, so the LDPC layer holds

$$\Gamma_{\rm LDPC} = 2n_L - k_L = 2(H-1)L + 2\Delta_{\rm LDPC} \tag{126}$$

physical qubits.

The repetition layer spans the whole length of the LDPC layer and consists of the following zones, from top to bottom in Figure 53. The *LDPC layer access zone*, of H rows of data qubits facing the block and its staircases, gives access to both logical operators of every logical qubit (Section C.1.3): X_L through a CNOT gate with the facing tip qubit, and Z_L through a repetition code covering its support (Section C.5.3). Under the work zone, it is completed by the proxy row used by the ADDER. The *buffer routing zone*, of height $h_{\rm routing,buf} = 4$ and length $l_{\rm routing,buf} = l_{\rm work} + 2 + p_r + 9(n_{\rm buf}^{(\rm ADDER)} - 1)$, where p_r is the right padding of the code (Table 20), the length of the widest point of the right staircase, carries the routing codes that connect the inputs of each injection or correction step of the ADDER to the associated buffers (Figure 47). The *buffer zone* holds $n_{\rm buf}^{(\rm ADDER)}$ ADDER buffers of nine repetition codes of distance $d_{\rm rep}$, hence $l_{\rm buf} = 9n_{\rm buf}^{(\rm ADDER)}$ columns. The first buffer lies below the compute zone, with its right edge aligned with the widest row of the access zone, and the others lie to its right (Figure 47). At $d_{\rm LDPC} = 12$ and 22, where $p_r = 4$, this alignment places the first buffer under the last five columns of the block, as required by the routing of the ADDER (Section C.5.3). At $d_{\rm LDPC} = 16$, where $p_r = 5$, the routing of the ADDER would also fit with the first buffer under the last five columns of the block, which would save one column of the buffer routing zone, i.e. 8 qubits. We keep this alignment for simplicity, so that our qubit counts are slightly conservative. As on the repetition-cat architecture, the auxiliary qubits of LOOKUPs and the PHASEUP workspace are stored in the buffer zone. Finally, as in Figure 50, $h_{\rm routing,fac}$ rows of factory routing lie above a *factory zone* of length $l_{\rm fac}$ that holds the factories of Section C.4.

Under the exponent zone, the only accesses during the computation are X_L accesses to at most two control tips at a time, for the AND gates of LOOKUPs and PHASEUPs, which are performed sequentially. They are routed through the two bottom rows of the access zone, the address-routing rows, of $\Gamma_{\rm addr} = 4l_{\rm exp}$ physical qubits. The rest of the access zone under the exponent zone,

$$\Gamma_{\rm QFT} = 2H\, l_{\rm exp} - l_{\rm exp} - \Gamma_{\rm addr}, \tag{127}$$

i.e. its $2H$ qubits per column, each row of data qubits counted with the ancilla row above it, minus the exposed ancilla row above its top row (Section C.7.1) and the address-routing rows, is used only by the Z-basis measurements of the final inverse quantum Fourier transform (Duration paragraph). In the color-changed convention, these measurements need Z_L, which, unlike on the repetition-cat architecture (Section C.7.1), is not reachable by a transversal destructive measurement.

Adding the zones of the repetition layer and the factories to the LDPC layer gives the total number of physical qubits,

$$\begin{aligned} n_{\rm total}^{(\rm LDPC)} = \Gamma_{\rm LDPC} &+ \Big[2H(L+p_r) - (L+p_r) \\ &+ 2(l_{\rm work}+2) + 2\Delta_{\rm L}\Big] \\ &+ 2h_{\rm routing,buf}\, l_{\rm routing,buf} + 2d_{\rm rep}\, l_{\rm buf} \\ &+ 2h_{\rm routing,fac}\, l_{\rm fac} - \max(l_{\rm fac} - l_{\rm buf}, 0) \\ &+ n_{\rm fac}, \end{aligned} \tag{128}$$

whose terms count six parts of the chip. The first term is the LDPC layer. The second term is the access zone, of $2H$ qubits per column over the L columns of the block and the p_r columns of its right end. This term subtracts the exposed ancilla row of these $L+p_r$ columns and adds the proxy row under the work zone and the left end of the block. The third term is the buffer routing zone. The fourth term is the buffer zone, of $2d_{\rm rep}$ qubits per column, i.e. $d_{\rm rep}$ data qubits each counted with the ancilla above it, as in Equation (91). The fifth term is the factory routing zone, of $2h_{\rm routing,fac}$ qubits per column. This term subtracts the exposed tops of its $l_{\rm fac} - l_{\rm buf}$ columns beyond the buffer zone, which no routing qubit uses. The sixth term is the factories. At the left end, under the exponent zone, the access zone only faces the $\Delta_{\rm L}$ qubits of the left staircase. At the right end, the ADDER compilation uses the full padded rectangle of Equation (91), so that the access zone covers a rectangle of width p_r over its H rows. We assume that the two staircases have the same number of qubits, $\Delta_{\rm L} = \Delta_{\rm LDPC}/2$. For RSA-2048, the parameters of Table 22 give $l_{\rm exp} = 640$, $l_{\rm mem} = 26$ and $l_{\rm comp} = 51$,

hence $l_{\text{work}} = 77$, $L = 717$ and $k_L = 1434$. With $h_{\text{routing,buf}} = 4$ and $l_{\text{buf}} = 9n_{\text{buf}}^{(\text{Adder})}$, Equation (128) becomes

$$\begin{aligned} n_{\text{total}}^{(\text{LDPC})} &= 2868H + 3\Delta_{\text{LDPC}} + (2H+7)\,p_r \\ &\quad + 18\,n_{\text{buf}}^{(\text{Adder})} d_{\text{rep}} + 81\,n_{\text{buf}}^{(\text{Adder})} - 1433 \\ &\quad + (2h_{\text{routing,fac}} - 1)\,l_{\text{fac}} + n_{\text{fac}}, \end{aligned} \tag{129}$$

for $l_{\text{fac}} \geq 9n_{\text{buf}}^{(\text{Adder})}$, with $p_r = 5$ for the code of Section C.8.2. The factory parameters $h_{\text{routing,fac}}$, l_{fac} and n_{fac} are given in the same section.

The global routing of the repetition layer is given by the factory routing zone, the address-routing rows and the routing qubits under the work zone, Γ_{work}, i.e.

$$\begin{aligned} \Gamma_{\text{global}} &= 2h_{\text{routing,fac}}\,l_{\text{fac}} - \max(l_{\text{fac}} - l_{\text{buf}}, 0) \\ &\quad + \Gamma_{\text{addr}} + \Gamma_{\text{work}}. \end{aligned} \tag{130}$$

Here,

$$\begin{aligned} \Gamma_{\text{work}} &= 2H(l_{\text{work}} + p_r) - (l_{\text{work}} + p_r) \\ &\quad + 2(l_{\text{work}} + 2) + 2\Delta_{\text{L}} \\ &\quad + 2h_{\text{routing,buf}}\,l_{\text{routing,buf}}, \end{aligned} \tag{131}$$

denotes the access zone under the work zone and at the right end of the block, minus its exposed ancilla row, plus the proxy row, the access zone at the left end of the block and the buffer routing zone. As on the repetition-cat architecture, the global routing is counted during the whole shot, whatever the current operation. The largest operation is an Adder, whose $9n_{\text{buf}}^{(\text{Adder})}$ buffer columns fill the buffer zone, so that Equations (128) to (131) give the identity

$$\max_s \Gamma_{\text{total}}(s) + \Gamma_{\text{global}} + \Gamma_{\text{QFT}} = n_{\text{total}}^{(\text{LDPC})}, \tag{132}$$

where $\Gamma_{\text{total}}(s) = \Gamma_{\text{LDPC}} + 2d_{\text{rep}}\,n_{\text{col}}(s) + n_{\text{fac}}$ is the physical qubit count of slice s. Section C.8.2 evaluates both sides.

AccumulateResidue keeps the arrangement of the repetition-cat layout (Section C.7.1), with the address register in the memory zone, left of the compute zone, but the buffers are at the bottom right of the compute zone (Figure 53). The accumulation qubits occupy the two tip rows of the compute zone, one every three columns in each row, which leaves room for the Adder†. The Lookup loads each qubit of its target register directly left of an accumulation qubit, and the Adder† writes its output in the remaining free slot of each triple. At its widest point, the Adder† therefore uses all $3(\kappa+1)$ slots of the compute zone and the $n_{\text{buf}}^{(\text{Adder})}$ Adder buffers. The output qubits of the Lookup are then uncomputed by the Z-basis measurement $M_Z(\kappa+1, 3)$ (Section C.8.1.iii). The Adder that follows runs on triples shifted by one slot, so the output of the last bit of the bottom row lands on the first slot of the top row, at the opposite end of the compute zone. This long-range output should be routable without additional qubits, although the routing is not demonstrated here. In the worst case, one additional compute column is needed, which adds $4H + 7$ physical qubits. After the Adder, the accumulation register sits two slots to the right of its initial position and moves back with Reshuffle$(\kappa, 3)$ (Section C.8.1.iii).

The other subroutines use the same operations, and Table 26 checks, as Table 23 does on the repetition-cat architecture, that every slice fits in the layout of Figure 53. For each operation, it counts the columns of the LDPC layer used in the memory and compute zones and the positions of the buffer zone, with three rules. A register laid out for an n-bit addition, one qubit per triple (a_k, b_k, s_k) in the two tip rows, spans $\lceil 3n/2 \rceil$ columns of the compute zone, whether only one summand is present, a Lookup has written the second one into the free slots of the triples, or the Adder fills all three slots. A register stored outside any addition is packed in $\lceil m/2 \rceil$ columns for m qubits. Finally, an Adder uses the $9n_{\text{buf}}^{(\text{Adder})}$ positions of its buffers, a Lookup$(w, \cdot)$ the $6n_{\text{buf}}^{(\text{Lookup})}$ positions of its buffers and its $w - 1$ auxiliary qubits, and a Phaseup(w) the same buffers as a Lookup and its $2^h - h + 2^l - l - 2$ auxiliary qubits corresponding to power products of weight at least two. These counts depend only on the parameters of Table 22 and on the numbers of buffers, not on the distances. Their maxima, 26 columns of memory zone, 51 columns of compute zone and 27 buffer positions, are all reached in AccumulateResidue, and they set l_{mem}, l_{comp} and l_{buf}. With the $l_{\text{exp}} = 640$ columns of the exponent zone, the block has $L = 717$ columns, which every slice of AccumulateResidue fills.

In ComputeResidue and ComputeResidue†, the accumulation register, packed in 17 columns, sits next to the widest operations, the Q-Q Adder$^\dagger(21)$ and the Q-Q Comparison(20), which take 32 columns ($17 + 32 = 49 \leq 51$), while $S_i \bmod (p_i - 1)$, the second ℓ-qubit register and the quotient register fill the 26 columns of the memory zone. The Q-Q Adder(31) of ComputeExponent takes 47 columns, and $17 + 47 = 64 > 51$. As on the repetition-cat architecture, the accumulation register is therefore parked in the memory zone during ReduceExponent†, ComputeExponent and ReduceExponent, next to the quotient register, which holds at most its $\log(n_e)$ qubits ($17 + 6 = 23 \leq 26$ columns), since ReduceExponent† empties it and ReduceExponent fills it. The register moves of Figure 52 therefore carry over to the LDPC-cat architecture. The buffer zone serves every Adder, since its $9n_{\text{buf}}^{(\text{Adder})}$ positions do not depend on the register size, and Lookups and Phaseups use at most 17 and 20 of its 27 positions.

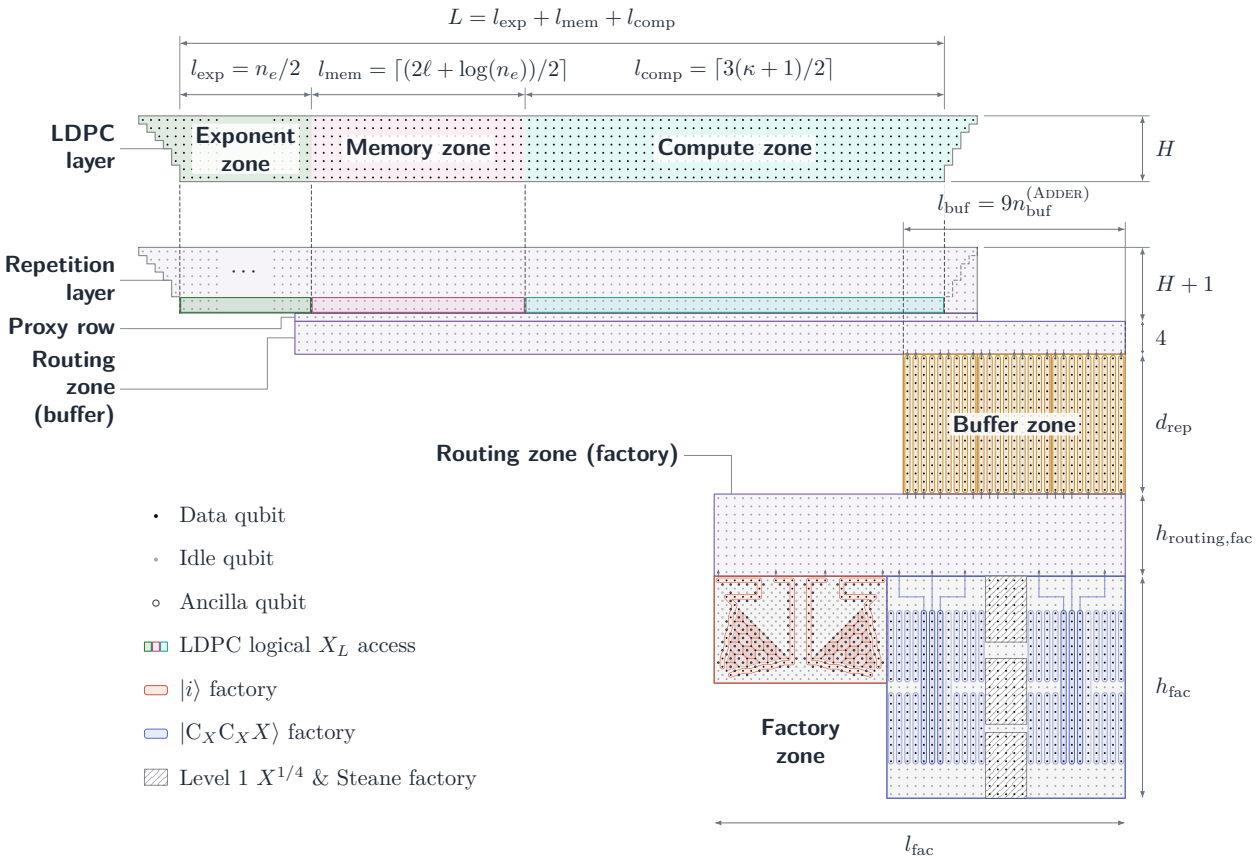


Figure 53: Global layout of the LDPC-cat chip for RSA-2048.

iii Access to Z_L: Z-basis measurements and register moves On the repetition-cat architecture, a Z-basis measurement $M_Z(x)$ is a transversal destructive measurement of one cycle (Section C.1.1), and registers move through the routing zone at a cost neglected in T_{shot} (Section C.7.1). In the LDPC layer, both operations need the operator Z_L of the logical qubits involved, since a move ends with a Z-basis measurement of its source. A Z-basis measurement of a logical qubit of the LDPC layer is performed by a snake repetition code of the access zone covering $\text{supp}(Z_L)$: the code is initialized in $|0\rangle_L$, receives transversal inter-layer Cnot gates from $\text{supp}(Z_L)$ during its first stabilizer round and is measured in the Z basis after $d^\star$ rounds, so that the measurement takes $T_{\text{surg}}^{(\text{LDPC})}$ cycles. Two logical qubits whose Z_L supports overlap cannot be measured simultaneously. Since a Z_L support spans at most $2H-1$ columns, each holding two logical qubits, measuring a register of n logical qubits that occupies one column out of v takes at most

$$T_{M_Z}(n, v) = \min\left(2\left\lceil \frac{2H-1}{v} \right\rceil, n\right) T_{\text{surg}}^{(\text{LDPC})} \quad (133)$$

cycles, with $T_{M_Z}(n) = T_{M_Z}(n, 1)$. The registers handled by the Adders, which occupy one column out of three (Section C.5.3), have $v = 3$.

A logical qubit moves from a slot a to a slot b of the LDPC layer by teleportation, with b in a Z_L eigenstate. The X-parity $X_L(a)X_L(b)$ is first measured between the tips of the two slots, through routing codes of the repetition layer, in $T_{\text{surg}}^{(\text{LDPC})}$ cycles. $Z_L(a)$ is then measured with a snake under the support of a only, as above. Slot b then holds the state of a up to Pauli corrections, which are absorbed in the Pauli frame, and slot a is left in a Z_L eigenstate, ready to receive a later move. The destination slots are always in a Z_L eigenstate, since they are freed either by a Z-basis measurement, such as the measurement-based uncomputation of the Lookup outputs or the consumption of an Adder input, or by an earlier move. The only access to Z_L is local, so that the overlap constraint of Equation (133) applies to the source register, while the long-range part of the move is an X-parity, like the other long-range operations of the architecture. The X-parities of several moves run in parallel on separate lanes, i.e. horizontal repetition codes of the repetition layer. When moving between the memory and compute zones, the lanes use the access zone, the proxy row and the buffer routing zone under the work zone, which hold 12 rows of data qubits. Since two neighboring lanes cannot share a row of ancillae, the 11 rows of ancillae between these rows of data qubits give at most 11 lanes.

		LDPC-layer columns			Buffer zone
Subroutine	Operation	Memory	Compute	Block	positions
ComputeExponent	Lookup$(6, 31)$	17	47	704	17
	Q-Q Adder(31)	17	47	704	27
ReduceExponent	Q-C Adder(21)	23	37	700	27
	Q-Q Adder(20)	23	36	699	27
ComputeResidue	Lookup$(6, 20)$ (loading)	16	47	703	17
	Lookup$(6, 21)$	26	49	715	17
	Q-Q Adder$^\dagger(21)$	26	49	715	27
	Q-Q Adder(20)	26	47	713	27
AccumulateResidue	Lookup$(5, 34)$	26	51	717	16
	Q-Q Adder$^\dagger(34)$	26	51	717	27
	Q-Q Adder(33)	26	51	717	27
	Lookup$(5, 33)$	26	51	717	16
	Q-Q Comparison(33)	26	51	717	27
	Phaseup(5)	26	51	717	17
ComputeResidue†	Lookup$(6, 21)$	26	49	715	17
	Q-Q Adder$^\dagger(21)$	26	49	715	27
	Q-Q Adder(20)	26	47	713	27
	Lookup$(6, 20)$	26	47	713	17
	Q-Q Comparison(20)	26	49	715	27
	Phaseup(6)	26	47	713	20
	Phaseup(6) (unload)	16	17	673	20
Last Phaseup	Phaseup(6)	17	0	657	20

Table 26: Layout of each slice on the LDPC-cat architecture for the factorization of an RSA-2048 number, with the parameters of Table 22, $n_{\mathrm{buf}}^{(\textsc{Adder})} = 3$ and $n_{\mathrm{buf}}^{(\textsc{Lookup})} = 2$. The counts do not depend on the distances. Memory and Compute are the numbers of columns of the LDPC layer, two logical qubits each, used in the memory and compute zones. A register laid out for an n-bit addition, one qubit per triple (a_k, b_k, s_k) in the two tip rows, spans $\lceil 3n/2 \rceil$ columns, whether only one summand is present, a Lookup has written the second one, or the Adder fills all three slots (a Comparison of size m is an Adder of size $m+1$). A register stored outside any addition is packed in $\lceil m/2 \rceil$ columns for m qubits. Block adds the $l_{\mathrm{exp}} = 640$ columns of the exponent zone, used during the whole computation. Buffer zone positions counts the repetition codes of the buffer zone used by the operation: $9n_{\mathrm{buf}}^{(\textsc{Adder})}$ for an Adder, $6n_{\mathrm{buf}}^{(\textsc{Lookup})} + w - 1$ for a Lookup$(w, \cdot)$, and $6n_{\mathrm{buf}}^{(\textsc{Lookup})} + 2^h - h + 2^l - l - 2$ for a Phaseup(w); it is the column count $n_{\mathrm{col}}(s)$ of Equation (143). The maxima, 26 memory columns, 51 compute columns, $L = 717$ columns of the block and 27 buffer positions, are all reached in AccumulateResidue and fix l_{mem}, l_{comp}, L and l_{buf} in the global layout of Figure 53. The table omits the M_Z, Reshuffle, PCX and $P_{|+\rangle}$ slices, which never exceed the adjacent Adder or Lookup slices.

We subtract one lane to account for the crossings of the vertical segments to the tips by simultaneous lattice surgery [46], which gives the estimate $n_{\mathrm{lane}} = 10$. A drift compensation moves each qubit by one or two slots only, so that all its X-parities run at once, $n_{\mathrm{lane}} = n$. Moving an n-qubit register that occupies one column out of v therefore takes

$$T_{\textsc{Reshuffle}}(n, v) = T_{M_Z}(n, v) + \left\lceil \frac{n}{n_{\mathrm{lane}}} \right\rceil T_{\mathrm{surg}}^{(\mathrm{LDPC})} \tag{134}$$

cycles, with $n_{\mathrm{lane}} = n$ for a drift compensation and $n_{\mathrm{lane}} = 10$ for a move between zones, marked "swap" in Table 27, where these operations appear as $M_Z(x, v)$ and Reshuffle(x, v). $T_{\textsc{Reshuffle}}(n, v)$ is an estimate that does not take into account the time ordering of X-parities and Z-measurements or the exact locations of the source and destination slots. An estimate is sufficient because this term is small (Section C.8.2.ii).

iv Duration All durations are counted in physical error-correction cycles. As in Section C.7.1, Equation (93) gives the duration of an n-bit Adder,

$$T_{\textsc{Adder}}^{(\mathrm{LDPC})}(n) = 2n\, T_{\mathrm{cad}}, \tag{135}$$

which also applies to the Q-C Adder and to a Comparison of size n, implemented as an Adder of size $n+1$. For $T_{\mathrm{cad}} = 21$, this is $42n$. In a Lookup, the And gates and the parity-Cnot gates are surgeries involving the LDPC layer. Equation (98) then gives

$$T_{\textsc{Lookup}}^{(\mathrm{LDPC})}(w) = (2^w - 1)\, T_{\mathrm{surg}}^{(\mathrm{LDPC})} + (2^{w-1} - w)\, T_{\mathrm{cad}}, \tag{136}$$

that is $63T_{\mathrm{surg}}^{(\mathrm{LDPC})} + 26T_{\mathrm{cad}}$ for $w_1 = 2w_3 = 6$ and $31T_{\mathrm{surg}}^{(\mathrm{LDPC})} + 11T_{\mathrm{cad}}$ for $w_4 = 5$. As in [47, Eqs. (4.8)

and (4.12)], this assumes that the factories produce a $|\mathrm{C}_X\mathrm{C}_X X\rangle$ state within $2T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ cycles, which holds for every pair of distances considered here ($21 \leq 2T_{\mathrm{surg}}^{(\mathrm{LDPC})}$, since $T_{\mathrm{surg}}^{(\mathrm{LDPC})} \geq 12$). Equation (101) gives the duration of a PHASEUP on a w-qubit address register, split into $h = \lceil w/2 \rceil$ and $l = \lfloor w/2 \rfloor$,

$$T_{\textsc{PhaseUp}}^{(\mathrm{LDPC})}(w) = (2^h + 2^l - w - 2)\, T_{\mathrm{cad}} + (2^l - 1)\, T_{\mathrm{surg}}^{(\mathrm{LDPC})}, \tag{137}$$

since its AND gates and multi-target $\mathrm{C}_X X$ gates are surgeries involving the LDPC layer, that is $8T_{\mathrm{cad}} + 7T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ for $w = 6$ and $5T_{\mathrm{cad}} + 3T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ for $w_4 = 5$. The slices of Table 15 omitted from Table 27 are, in the color-changed convention, the parity-CNOT gates $\mathrm{PC}X(x)$ and the preparations $P_{|+\rangle}(x)$. A $\mathrm{PC}X(x)$ is a single lattice surgery (Section C.7.1), and takes $T_{\mathrm{PC}X}^{(\mathrm{LDPC})} = T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ cycles. A preparation of logical qubits of the LDPC layer takes $T_{\mathrm{prep}}^{(\mathrm{LDPC})} = T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ cycles, instead of $T_{\mathrm{init}} = 1$ cycle on the repetition-cat architecture. A logical qubit of the LDPC layer shares its physical qubits with other logical qubits and cannot be reinitialized at the physical level. The qubit is therefore prepared in $|+\rangle_L$ by measuring its weight-one X_L through the facing qubit of its tip over $d^\star$ rounds, and the sign of the outcome is absorbed in the Pauli frame. The $M_Z(x, v)$ and $\textsc{Reshuffle}(x, v)$ slices last $T_{M_Z}(x, v)$ cycles (Equations (133) and (134)). The Z-basis measurements are the N_{M_Z} slices of Equation (113), which appear here as rows of Table 27.

Registers drift and move within subroutines as on the repetition-cat architecture (Section C.7.1). Unlike there, these moves are included in T_{shot}, as RESHUFFLE slices of Table 27. Each iteration of COMPUTEEXPONENT adds $T_{\textsc{Reshuffle}}(\ell + \log(n_e), 3)$, each iteration of ACCUMULATERESIDUE adds $T_{\textsc{Reshuffle}}(\kappa, 3)$, and each iteration of REDUCEEXPONENT and REDUCEEXPONENT† adds $T_{\textsc{Reshuffle}}(\ell + 1, 3) + T_{\textsc{Reshuffle}}(2, 1)$, the second term moving the underflow qubit and the next bit of S_i, that is $3T_{\mathrm{surg}}^{(\mathrm{LDPC})}$. In COMPUTERESIDUE, the drift of each inner-loop iteration is compensated by a $\textsc{Reshuffle}(\ell, 3)$. At the end of each outer-loop iteration, this move is replaced by the move of the newly computed register to the memory slots freed by the Z-basis measurement of the former register, a move between zones. In COMPUTERESIDUE†, no measurement frees the memory slots, so the register they hold first moves into the compute zone, in the slots freed by the drift of the ADDER, with one $\textsc{Reshuffle}(\ell, 1)$ per outer-loop iteration. The newly computed register then moves to the memory slots with a $\textsc{Reshuffle}(\ell, 3)$, again in place of the drift compensation. With $N_{\mathrm{out}} = |P|(\lceil \ell/w_3 \rceil - 2)$ outer-loop iterations, $c = \lceil \ell/w_3 \rceil$ inner-loop iterations per outer iteration in COMPUTERESIDUE and $2c$ in COMPUTERESIDUE†, and $T_{\textsc{Reshuffle}}^{\mathrm{swap}}$ denoting a move between zones, the moves within COMPUTERESIDUE and COMPUTERESIDUE† take

$$N_{\mathrm{out}} \Big[(c-1)\, T_{\textsc{Reshuffle}}(\ell, 3) + T_{\textsc{Reshuffle}}^{\mathrm{swap}}(\ell, 3) \Big] \quad \text{and}$$
$$N_{\mathrm{out}} \Big[(2c-1)\, T_{\textsc{Reshuffle}}(\ell, 3) + T_{\textsc{Reshuffle}}^{\mathrm{swap}}(\ell, 3) + T_{\textsc{Reshuffle}}^{\mathrm{swap}}(\ell, 1) \Big] \tag{138}$$

cycles, respectively.

Summing over the numbers of calls N_{op} of Table 27, which depend only on Table 22 and are those of Table 25. As in Equation (110), the duration of one shot is

$$T_{\mathrm{shot}} = \sum_{\mathrm{op}} N_{\mathrm{op}}\, T_{\mathrm{op}} + T_{\mathrm{other}}, \tag{139}$$

where the sum runs over the operations of Table 27, including the M_Z and RESHUFFLE slices, with the durations of Equations (133) to (138), and T_{other} collects the parity-CNOT and preparation slices,

$$T_{\mathrm{other}} = (N_{\mathrm{PC}X} + N_{\mathrm{prep}})\, T_{\mathrm{surg}}^{(\mathrm{LDPC})}, \tag{140}$$

with $N_{\mathrm{PC}X}$ and N_{prep} given by Equations (112) and (114).

As on the repetition-cat architecture, the moves between subroutines are not included in T_{shot}. They follow Figure 52: per prime, the accumulation register moves four times, once from its interleaved layout ($v = 3$) and three times packed ($v = 1$), $S_i \bmod (p_i - 1)$ moves twice, packed, and r_i moves once from a layout for an addition ($v = 3$), while REDUCEEXPONENT†, COMPUTEEXPONENT and REDUCEEXPONENT are entered and left without moves. All these moves are between zones, so that

$$T_{\mathrm{transit}} = |P| \Big[3\, T_{\textsc{Reshuffle}}^{\mathrm{swap}}(\kappa, 1) + T_{\textsc{Reshuffle}}^{\mathrm{swap}}(\kappa, 3) + 2\, T_{\textsc{Reshuffle}}^{\mathrm{swap}}(\ell, 1) + T_{\textsc{Reshuffle}}^{\mathrm{swap}}(\ell, 3) \Big] \tag{141}$$

cycles, whose share of T_{shot} is given in Section C.8.2. The final inverse quantum Fourier transform is not included in T_{shot} either. As in Section C.7.1, each of the n_e exponent qubits receives one rotation by phase kickback into a phase-gradient register of n_φ qubits [104], and is then measured in the Z basis, here through the access zone under the exponent zone, in $T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ cycles. The transform takes

$$T_{\mathrm{QFT}} = n_e \Big[T_{\textsc{Adder}}^{(\mathrm{LDPC})}(n_\varphi) + T_{\mathrm{prep}}^{(\mathrm{LDPC})} + T_{\mathrm{PC}X}^{(\mathrm{LDPC})} + T_{\mathrm{surg}}^{(\mathrm{LDPC})} + T_{\mathrm{react}} \Big] \tag{142}$$

cycles, and $n_e(n_\varphi - 1)$ Toffoli gates.

v Volume The logical error rate of the repetition code is defined per d_{rep} cycles (Equation (122), from [16, Supplementary Table 1(b)]), and that of the LDPC-cat code per d_{LDPC} cycles [47, Eq. (6.1)]. A

physical duration T therefore counts as T/d_{rep} logical cycles in the repetition layer and as T/d_{LDPC} logical cycles in the LDPC layer. A repetition code that takes part in a surgery of $T_{\text{surg}}^{(\text{LDPC})}$ cycles is exposed for $T_{\text{surg}}^{(\text{LDPC})}/d_{\text{rep}}$ logical cycles, and the volume accounts for this exposure automatically.

For the repetition layer, Equations (115) and (116) become

$$\text{Vol}_{\text{physical}}^{(\text{rep})} = \sum_s \Big[2d_{\text{rep}}\, n_{\text{col}}(s) + \Gamma_{\text{global}}\Big] \times T_{\text{error correction}}(s), \tag{143}$$

$$\text{Vol}_{\text{logical}}^{(\text{eff,rep})} = \frac{\text{Vol}_{\text{physical}}^{(\text{rep})}}{d_{\text{rep}}(2d_{\text{rep}}-1)}. \tag{144}$$

For the LDPC layer, counted as allocated during the whole shot,

$$\text{Vol}_{\text{physical}}^{(\text{LDPC})} = \Gamma_{\text{LDPC}}\, T_{\text{shot}}. \tag{145}$$

The LDPC layer encodes k_L logical qubits on Γ_{LDPC} physical qubits, so the factor $2d-1$ of Equation (116) becomes Γ_{LDPC}/k_L,

$$\text{Vol}_{\text{logical}}^{(\text{eff,LDPC})} = \frac{k_L\, \text{Vol}_{\text{physical}}^{(\text{LDPC})}}{\Gamma_{\text{LDPC}}\, d_{\text{LDPC}}} = k_L\, \frac{T_{\text{shot}}}{d_{\text{LDPC}}}, \tag{146}$$

and the effective number of logical qubits of the LDPC layer is k_L by construction. The logical error rate per logical cycle of one logical qubit of the repetition layer, $p_{\text{logical}}^{(\text{rep})}$, is given by Equation (122) at $d = d_{\text{rep}}$. For the LDPC layer, we use the fit of [47, Eq. (6.1)], obtained under the same phase-flip circuit-level noise model,

$$p_{\text{logical}}^{(\text{LDPC})}(p_Z, d_{\text{LDPC}}) = A\, d_{\text{LDPC}}\, (B\, p_Z)^{C\lfloor (d_{\text{LDPC}}+1)/2 \rfloor}, \tag{147}$$

with $A = 0.1$, $C = 1$ and $B = 25.8$, 29.8 and 42.5 for $d_{\text{LDPC}} = 12$, 16 and 22 [47, Table XII]. Equation (147) is a per-logical-qubit rate. We assume that logical errors occur independently in the two layers. With $p_{\text{logical}}^{(\text{rep})}$ and $p_{\text{logical}}^{(\text{LDPC})}$, the probability that at least one logical error occurs during one shot is, as in Equation (117),

$$\epsilon_{\text{logical}}^{(j)} = 1 - \Big(1 - p_{\text{logical}}^{(j)}\Big)^{\text{Vol}_{\text{logical}}^{(\text{eff},j)}}, \quad j \in \{\text{rep}, \text{LDPC}\}, \tag{148}$$

in each layer, and

$$\epsilon_{\text{total}} = 1 - \Big(1 - \epsilon_{\text{logical}}^{(\text{rep})}\Big)\Big(1 - \epsilon_{\text{logical}}^{(\text{LDPC})}\Big) \tag{149}$$

in either layer.

As in Section C.7.1, these volumes rest on a few assumptions. First, the whole LDPC layer is allocated during the whole shot. The exponent register alone occupies $n_e = 1280$ of the $k_L = 1434$ slots, so at most 154 slots are unused at any time, and the LDPC volume is overestimated by at most $k_L/n_e - 1 \approx 12\%$. Second, the slices omitted from Table 26 (M_Z, RESHUFFLE, PCX and $P_{|+\rangle}$) are assigned the peak column count of the computation, $n_{\text{col,other}} = 9n_{\text{buf}}^{(\text{ADDER})}$, an upper bound since these slices are always smaller than the adjacent ADDERs or LOOKUPs. Third, as in Equation (103), the routing qubits are counted at all times, although an operation only uses part of them. In the access zone and the routing zone under the work zone, a surgery uses only the snakes and routing codes it needs. The address-routing rows Γ_{addr} are only used during the LOOKUPs of COMPUTEEXPONENT and the last PHASEUP, about 27% of T_{shot}, and each access uses only the part of these rows between the control tip and the end of the exponent zone. Γ_{QFT}, used only during the final transform, enters the physical count but not the volume. Fourth, REDUCEEXPONENT† has the same footprints and durations as REDUCEEXPONENT.

C.8.2 Evaluation at $p_Z = 5 \times 10^{-4}$ and $p_Z = 10^{-3}$

We now evaluate the estimate for RSA-2048 in detail at $p_Z = 5 \times 10^{-4}$, the phase-flip probability of the main result of Section C.7.2, and give its main figures at $p_Z = 10^{-3}$. Table 28 gives the logical error budget at both values of p_Z and for other pairs of distances.

i Factories, distances and physical qubit count We use the factories of Table 19, as in Section C.7.2, with outputs at distance d_{rep}. We choose $(d_{\text{rep}}, d_{\text{LDPC}}) = (13, 16)$, hence $d^\star = T_{\text{surg}}^{(\text{LDPC})} = 16$ (Equation (78)), for the reasons given in the logical error budget below. Equation (123) gives $T_{\text{cad}} = 21$, set by the $|\text{C}_X\text{C}_X X\rangle$ factory, and Equation (82) one $|\text{C}_X\text{C}_X X\rangle$ factory and two $|i\rangle$ factories. As in Section C.7.2, the factory zone has length $l_{\text{fac}} = 50$ and contains $n_{\text{fac}} = 1872$ physical qubits, and the factory routing zone has height $h_{\text{routing,fac}} = 10$, hence $2h_{\text{routing,fac}}l_{\text{fac}} - (l_{\text{fac}} - l_{\text{buf}}) = 1000 - 23 = 977$. Equation (124) gives $T_{\text{buf}} = 49$ cycles, hence $n_{\text{buf}}^{(\text{ADDER})} = 3$. The code of Table 20 with $d_{\text{LDPC}} = 16$ has $H = 7$ and $\Delta_{\text{LDPC}} = 26$, hence $n_L = 5045$, $k_L = 1434$ and $\Gamma_{\text{LDPC}} = 8656$ (Equation (126)).

Since all logical data qubits lie in the LDPC layer, each slice s adds to the Γ_{LDPC} qubits of the LDPC layer and to the factories only the $2d_{\text{rep}}\, n_{\text{col}}(s)$ physical qubits of its buffer positions (Buffer zone column of Table 26). The maximum, $8656 + 27 \times 26 + 1872 = 11\,230$ physical qubits, is reached by every ADDER slice.

Equation (129) gives $n_{\text{total}}^{(\text{LDPC})} = 22\,593$ physical qubits. The global routing of Equation (130) is $\Gamma_{\text{global}} = 5603$, whose terms are 977 (factory routing zone), $\Gamma_{\text{addr}} = 2560$ and $\Gamma_{\text{work}} = 2066$ (Equation (131)): 1001 for the access zone under the work zone, 158 for the proxy row, 26 for the left staircase ($\Delta_{\text{L}} = 13$), 65 for the rectangle of width $p_r = 5$ at the right end, and 816 for the buffer routing zone. With $\Gamma_{\text{QFT}} = 5760$ (Equation (127)), these values satisfy

the identity of Equation (132): $11\,230+5603+5760 = 22\,593$. Each exponent logical qubit costs 12.5 physical qubits. The LDPC layer accounts for 38% of the chip, and Γ_{QFT}, idle until the final transform, for 25%. A destructive Z-basis measurement of all the physical qubits of the LDPC layer would give Z_L of every exponent qubit in a single cycle, without any access zone. The semi-classical transform, however, measures the exponent qubits one after the other, the rotation of each qubit depending on the outcomes of the qubits already measured, so that it cannot use this simultaneous measurement as such. Avoiding Γ_{QFT}, or using its qubits for other purposes during the computation, therefore requires another way of accessing Z_L of the exponent qubits, or a transform compatible with a simultaneous measurement. We leave this investigation for future work.

ii Duration Table 27 gives the duration of each slice, in the format of Table 25. For RSA-2048, $N_{\mathrm{PC}X} = 262|P|$ and $N_{\mathrm{prep}} = 5|P|$ (Section C.7.1), so that Equation (140) gives $T_{\mathrm{other}} = 267|P| \times 16 \approx 9.7 \times 10^7$ cycles, 0.34% of T_{shot}, and the $N_{M_Z} = 400|P| + 214$ Z-basis measurements of Table 27 take 1.34×10^9 cycles. Equation (139) then gives $T_{\mathrm{shot}} = 2.82 \times 10^{10}$ cycles per shot, about 7.84 hours. Adders and Comparisons take 43.53% of the duration, Lookups 44.63%, Phaseups 1.65%, Z-basis measurements 4.73% and register moves within subroutines 5.11% (1.44×10^9 cycles, about 8.4×10^6 moves), above the 0.5% threshold of Section C.7.1. The X-parity term of Equation (134) accounts for 1.4×10^8 of these cycles, 0.50% of T_{shot}. For example, measuring the accumulation register takes $T_{M_Z}(\kappa, 3) = 160$ cycles (Equation (133)). The Z-basis measurements and the moves, which are negligible on the repetition-cat architecture, thus take 9.8% of a shot. The moves between subroutines, about 1.6×10^5 per shot, take 2560 cycles per prime, that is 5.8×10^7 cycles (Equation (141)), 0.21% of T_{shot}, below this threshold, and are neglected, as on the repetition-cat architecture. The final inverse transform takes $T_{\mathrm{QFT}} = 1280 \times 1108 \approx 1.42 \times 10^6$ cycles for $n_\varphi = 25$ (Equation (142)), that is 5×10^{-5} of T_{shot}, and 3.1×10^4 Toffoli gates, and is neglected as well.

iii Logical error budget, number of shots and total runtime Tables 26 and 27 with Equations (143) to (146) give $\mathrm{Vol}^{(\mathrm{eff,rep})}_{\mathrm{logical}} = 5.4 \times 10^{11}$ and $\mathrm{Vol}^{(\mathrm{eff,LDPC})}_{\mathrm{logical}} = 2.5 \times 10^{12}$. The global routing Γ_{global} accounts for 91% of the volume of the repetition layer, the address-routing rows Γ_{addr} alone for 41%, and the repetition-code columns of the operations for 9%. The logical error rate per logical cycle of the repetition layer is $p^{(\mathrm{rep})}_{\mathrm{logical}} = 5.67 \times 10^{-14}$ (Equation (122) at $d = 13$, Table 24), and that of the LDPC layer is $p^{(\mathrm{LDPC})}_{\mathrm{logical}} = 3.9 \times 10^{-15}$ (Equation (147)). Equations (148) and (149) then give $\epsilon^{(\mathrm{rep})}_{\mathrm{logical}} = 3.0 \times 10^{-2}$, $\epsilon^{(\mathrm{LDPC})}_{\mathrm{logical}} = 9.8 \times 10^{-3}$ and $\epsilon_{\mathrm{total}} = 4.0 \times 10^{-2}$. With the factories of Table 19, and counting Toffoli gates and $|i\rangle$ states as in Section C.7.2, the $|\mathrm{C}_X\mathrm{C}_X X\rangle$ and $|i\rangle$ factories cause a failure during a shot with probabilities 1.5×10^{-4} and 0.20%, as on the repetition-cat architecture. The per-shot failure probability is therefore $\epsilon = 1-(1-\epsilon_{\mathrm{total}})(1-\epsilon_{|\mathrm{C}_X\mathrm{C}_X X\rangle})(1-\epsilon_{|i\rangle}) = 4.2\%$, dominated by the repetition layer.

For $s = 8$, the average number of shots is 9.2 (Equation (60)). Accounting for the per-shot failure probability ϵ, it becomes $9.2/(1-\epsilon) = 9.60$ at $p_Z = 5 \times 10^{-4}$, which brings the total duration to 9.60×7.84 h ≈ 75.3 h ≈ 3.14 days.

We apply the criterion of Section C.7.2.ii to the pairs $(d_{\mathrm{rep}}, d_{\mathrm{LDPC}})$: among the pairs for which ϵ is below 10%, we choose the one with the fewest physical qubits $n^{(\mathrm{LDPC})}_{\mathrm{total}}$ (Equation (128)). Table 28 gives the volumes and the logical error probabilities per shot for several pairs, at $p_Z = 10^{-3}$ and $p_Z = 5 \times 10^{-4}$. At both values of p_Z, with $d_{\mathrm{LDPC}} = 12$, at least one logical error occurs in the LDPC layer in almost every shot, and with $d_{\mathrm{rep}} = 11$ in the repetition layer. At $p_Z = 5 \times 10^{-4}$, the criterion selects $(13, 16)$, with 22 593 physical qubits, against 22 701 for the next pair with $\epsilon < 10\%$, $(15, 16)$. The pair $(15, 16)$ would divide ϵ by about 3.4, to 1.2%, for $2 \times 18\, n^{(\textsc{Adder})}_{\mathrm{buf}} = 108$ more physical qubits (Equation (129)). Since its surgeries also last $T^{(\mathrm{LDPC})}_{\mathrm{surg}} = d_{\mathrm{LDPC}} = 16$ cycles (Equation (78)), its shots have the same duration, and its 3.0% fewer shots would shorten the total runtime from 3.14 to 3.04 days. At $p_Z = 10^{-3}$, the factories add about 1.8% to $\epsilon_{\mathrm{total}}$ (Section C.7.2), and the pair $(15, 22)$ gives $\epsilon = 10.9\%$, just above the threshold, since its surgeries last $T^{(\mathrm{LDPC})}_{\mathrm{surg}} = 22$ cycles, which also sets $T_{\mathrm{cad}} = 22$. Only $(17, 22)$ and $(19, 22)$ then have $\epsilon < 10\%$, and the criterion selects $(17, 22)$, with $\epsilon = 2.3\%$, whose repetition distance coincides with the distance $d = 17$ selected at this p_Z in Section C.7.2. It uses 25 673 physical qubits, against 25 781 for $(19, 22)$, and its shots last 3.32×10^{10} cycles, about 9.24 hours, for a total runtime of 3.63 days.

iv Comparison with the repetition-cat architecture Table 29 compares the two architectures at $p_Z = 5 \times 10^{-4}$ and $p_Z = 10^{-3}$. At $p_Z = 5 \times 10^{-4}$, the LDPC-cat architecture reduces the physical qubit count by a factor of 2.2, from 49 649 to 22 593. The exponent register accounts for 88% of this reduction. An exponent logical qubit costs 12.5 physical qubits on the LDPC-cat architecture, $\Gamma_{\mathrm{LDPC}}/k_L = 6.0$ in the LDPC layer and $(2H-1)/2 = 6.5$ in the access zone below it, whose $2H$ qubits per column, minus the exposed ancilla row, serve the two logical qubits of the column. On the repetition-cat architecture, it costs 31.1, that is $2d-1 = 29$ for its repetition code and 2.1 for its share of the routing zone between the two ex-

ponent zones (Equation (103)). In exchange, one shot is 13% longer, 2.82×10^{10} against 2.493×10^{10} cycles. The Z-basis measurements and register moves of Section C.8.1.iii add 2.8×10^9 cycles, whereas the ADDERs last $2nT_{\text{cad}}$ on both architectures (Equations (108) and (135)). The LOOKUPs follow the same schedule on both architectures (Equations (97) and (136)), but their $2^w - 1$ surgeries last 16 cycles instead of 15, so that a LOOKUP(6) lasts 1554 against 1491 cycles, which adds 5.1×10^8 cycles. The per-shot failure probability rises from 0.43% to 4.2%, which adds 3.9% to the number of shots, and the total runtime grows by 18%, from 2.67 to 3.14 days. The space-time cost of the computation, in physical qubit-days, is therefore divided by about 1.9. At $p_Z = 10^{-3}$, with $(17, 22)$ against $d = 17$, the physical qubit count is again divided by 2.2, from 55 497 to 25 673. One shot, however, is 28% longer, 3.32×10^{10} against 2.598×10^{10} cycles, since $T_{\text{surg}}^{(\text{LDPC})} = d^\star = 22$ sets both the surgery duration and the cadence $T_{\text{cad}} = 22$. The total runtime grows by 27%, from 2.85 to 3.63 days, so that the space-time cost is divided by 1.7.

C.9 Physical cat qubits and bit-flip assumptions

C.9.1 Cat qubits: encoding, stabilization, operations, and bit flips

In a cat qubit, the information is stored in a superconducting microwave resonator [19]. The computational states of the cat qubit are $|0\rangle \simeq |\alpha\rangle$ and $|1\rangle \simeq |-\alpha\rangle$, where the coherent states $|\pm\alpha\rangle$ are eigenstates of the annihilation operator $\hat{a}$ of the resonator mode, $\hat{a}\,|\pm\alpha\rangle = \pm\alpha\,|\pm\alpha\rangle$, and $\bar{n} = |\alpha|^2$ is their mean photon number. The dominant error channel of the resonator is single-photon loss, with jump operator $\hat{a}$ and rate κ_1.[17] In the basis $\{|\alpha\rangle, |-\alpha\rangle\}$, $\hat{a}$ acts as a phase flip, $\hat{a}\,(a_0\,|\alpha\rangle + a_1\,|-\alpha\rangle) = \alpha\,(a_0\,|\alpha\rangle - a_1\,|-\alpha\rangle)$, while a bit flip requires a transition between two states separated by $2|\alpha|$ in phase space. Once the manifold spanned by $|\pm\alpha\rangle$ is stabilized (see below), the phase-flip and bit-flip rates scale as

$$\Gamma_X \simeq 2\kappa_1\bar{n}, \qquad \Gamma_Z \propto e^{-c\,\bar{n}}, \tag{150}$$

where $c > 0$. Here, the subscript of Γ refers to the observable whose expectation value decays, $\langle X\rangle$ for phase flips and $\langle Z\rangle$ for bit flips. In contrast, for the Pauli error probabilities used in the rest of this paper, the subscript denotes the Pauli error itself, and p_X and p_Z refer to bit flips and phase flips, respectively. The factor 2 in Γ_X arises because photon losses occur at the rate $\kappa_1\bar{n}$ and each of them changes the sign of $\langle X\rangle$, which then decays as $e^{-2\kappa_1\bar{n}t}$. We call $T_Z = 1/\Gamma_Z$ the bit-flip time. After a time t, a qubit prepared in $|0\rangle$ is found in $|1\rangle$ with probability $(1 - e^{-t/T_Z})/2$, about $t/(2T_Z)$ for $t \ll T_Z$. The bit-flip rate is thus exponentially suppressed with $\bar{n}$, while the phase-flip rate increases only linearly [19, 26, 32]. With such a noise bias, only phase flips need to be corrected actively, which we do with repetition codes [32] and LDPC-cat codes [16].

The cat qubit manifold is stabilized by an engineered two-photon dissipation [19, 20, 26]. The resonator is coupled, through a nonlinear element, to a strongly dissipative mode, the buffer. This nonlinear element converts pairs of resonator photons into single buffer photons, and conversely. After adiabatic elimination of the buffer, the resonator undergoes the two-photon dissipation $\mathcal{D}[\hat{a}^2 - \alpha^2]$ at a rate κ_2 [19, 20, 105], where the α^2 term comes from a resonant drive on the buffer [26, 29]. The steady states of this dissipation are the coherent states $|\pm\alpha\rangle$ and their superpositions, and any state of the resonator relaxes to the cat qubit manifold and remains in it. The mean photon number $\bar{n}$, and with it the noise bias, is set by the amplitude of the buffer drive and can therefore be tuned in situ.

The physical operations of Section C.1.1 have bias-preserving implementations [4, 32]. These operations are the preparations in $|0\rangle$ and $|+\rangle$, the CNOT gates between neighboring qubits, and the measurements in the X and Z bases. All the logical operations of this paper are built from these physical operations, including the lattice surgery on repetition codes [4, 32] and the computing layer of the LDPC-cat codes [16]. The only exceptions are the factories that prepare the resource states $|\mathrm{C}_X\mathrm{C}_X X\rangle$ and $|i\rangle$ [47, 98], which also require rotations by $\pi/2$ and $\pi/4$ around the X axis of the Bloch sphere. These rotations do not need to be bias-preserving, since the bit flips they induce can be detected in the factory [98]. A rotation by $\pi/2$ around the X axis has been realized experimentally with a holonomic gate [106], as part of the tomography protocol of [29].

As the physical realizations of the gates on cat qubits evolve quickly, it is simpler to use a generic phase-flip noise model, against which a specific physical implementation can then be compared. Indeed, many gate designs have been proposed over the years [19, 32–34, 107–110], and recent proposals include CNOT gates between a cat qubit and a transmon [35] and between two cat qubits [111]. The CNOT gates of the syndrome extraction can even be replaced by quantum non-demolition measurements of multi-qubit Z operators [112, 113]. We use the circuit-level model of [16, Supplementary Table 1(b)], recalled in Table 30, in which every noisy operation has the same phase-flip probability p_Z. The correspondence between p_Z and the hardware parameters is discussed in [16, Methods and Supplementary Table 1(c)].

In theory, the bit-flip time T_Z of an idle cat qubit grows exponentially with $\bar{n}$ (Equation (150)). During a computation, however, the circuit-level models of [4,

[17] Elsewhere in this paper, κ without subscript is the truncation length of Appendix A, that is, the size of the accumulation register.

Subroutine	Operation	Calls	Cycles/call	Cycles	Share
ComputeExponent	Lookup$(6, 31)$	4 849 026	1554	7.54×10^9	26.70%
	Q-Q Adder(31)	4 849 026	1302	6.31×10^9	22.37%
	$M_Z(31, 3)$	4 849 026	160	7.76×10^8	2.75%
	Reshuffle$(31, 3)$	4 849 026	176	8.53×10^8	3.02%
ReduceExponent	Q-C Adder(21)	249 238	882	2.20×10^8	0.78%
	Q-Q Adder(20)	249 238	840	2.09×10^8	0.74%
	Reshuffle$(21, 3)$	249 238	176	4.39×10^7	0.16%
	Reshuffle$(2, 1)$	249 238	48	1.20×10^7	0.04%
ComputeResidue	Lookup$(6, 20)$ (loading)	22 658	1554	3.52×10^7	0.12%
	Lookup$(6, 21)$	793 030	1554	1.23×10^9	4.37%
	Q-Q Adder$^\dagger(21)$	793 030	882	6.99×10^8	2.48%
	$M_Z(21, 3)$	793 030	160	1.27×10^8	0.45%
	Q-Q Adder(20)	793 030	840	6.66×10^8	2.36%
	$M_Z(1)$	793 030	16	1.27×10^7	0.04%
	Reshuffle$(20, 3)$	679 740	176	1.20×10^8	0.42%
	Reshuffle$(20, 3)$ (swap)	113 290	192	2.18×10^7	0.08%
	$M_Z(20, 3)$	113 290	160	1.81×10^7	0.06%
AccumulateResidue	Lookup$(5, 34)$	90 632	727	6.59×10^7	0.23%
	Q-Q Adder$^\dagger(34)$	90 632	1428	1.29×10^8	0.46%
	$M_Z(34, 3)$	90 632	160	1.45×10^7	0.05%
	Q-Q Adder(33)	90 632	1386	1.26×10^8	0.45%
	Lookup$(5, 33)$	45 316	727	3.29×10^7	0.12%
	Q-Q Comparison(33)	45 316	1428	6.47×10^7	0.23%
	$M_Z(33, 3)$	45 316	160	7.25×10^6	0.03%
	Phaseup(5)	90 632	153	1.39×10^7	0.05%
	Reshuffle$(33, 3)$	90 632	176	1.60×10^7	0.06%
ComputeResidue†	Lookup$(6, 21)$	1 586 060	1554	2.46×10^9	8.73%
	Q-Q Adder$^\dagger(21)$	1 586 060	882	1.40×10^9	4.96%
	$M_Z(21, 3)$	1 586 060	160	2.54×10^8	0.90%
	Q-Q Adder(20)	1 586 060	840	1.33×10^9	4.72%
	Lookup$(6, 20)$	793 030	1554	1.23×10^9	4.37%
	Q-Q Comparison(20)	793 030	882	6.99×10^8	2.48%
	$M_Z(20, 3)$	793 030	160	1.27×10^8	0.45%
	Phaseup(6)	1 586 060	280	4.44×10^8	1.57%
	Reshuffle$(20, 1)$ (swap)	113 290	352	3.99×10^7	0.14%
	Reshuffle$(20, 3)$	1 472 770	176	2.59×10^8	0.92%
	Reshuffle$(20, 3)$ (swap)	113 290	192	2.18×10^7	0.08%
	Phaseup(6) (unload)	22 658	280	6.34×10^6	0.02%
ReduceExponent†	Q-C Adder(21)	249 238	882	2.20×10^8	0.78%
	Q-Q Adder(20)	249 238	840	2.09×10^8	0.74%
	Reshuffle$(21, 3)$	249 238	176	4.39×10^7	0.16%
	Reshuffle$(2, 1)$	249 238	48	1.20×10^7	0.04%
Last Phaseup	Phaseup(6)	214	280	5.99×10^4	0.00%

Table 27: Breakdown of cycle counts by operation for the factorization of an RSA-2048 number on the LDPC-cat architecture, for $p_Z = 5 \times 10^{-4}$ and $(d_{\text{rep}}, d_{\text{LDPC}}) = (13, 16)$, in the format of Table 25. The total duration is $T_{\text{shot}} = 2.82 \times 10^{10}$ cycles. Lattice surgeries involving the LDPC layer last $T_{\text{surg}}^{(\text{LDPC})} = 16$ cycles, loadings of factory outputs $d_{\text{rep}} = 13$ cycles, and Adders run at the cadence $T_{\text{cad}} = 21$ (Equations (135) to (137)). $M_Z(x, v)$ is the Z-basis measurement of x logical qubits of the LDPC layer, and Reshuffle(x, v) a move of x logical qubits within the LDPC layer: drift compensation after the Adders, role swap of the two ℓ-qubit registers, and moves of the underflow qubit and of the next bit of S_i in ReduceExponent. In both cases, the qubits occupy one column out of v, and $M_Z(x)$ stands for $M_Z(x, 1)$. $M_Z(x, v)$ lasts $T_{M_Z}(x, v)$ cycles and Reshuffle(x, v) lasts $T_{\text{Reshuffle}}(x, v)$ cycles (Section C.8.1.iii), with all X-parities in parallel for a drift compensation and $n_{\text{lane}} = 10$ lanes for a move between zones, marked "swap". For clarity, the table omits the $\text{PC}X$ and $P_{|+\rangle}$ slices, which contribute 0.34% of T_{shot} and are included in the total duration and in the volumes. The moves between subroutines (0.21% of T_{shot}) and the final inverse quantum Fourier transform are not included. Cycles/call refers to the number of cycles of one call of an operation, and Cycles to the total number of cycles of all its calls.

d_{rep}	d_{LDPC}	$\text{Vol}_{\text{logical}}^{(\text{eff,rep})}$	$\text{Vol}_{\text{logical}}^{(\text{eff,LDPC})}$	$\epsilon_{\text{logical}}^{(\text{rep})}$	$\epsilon_{\text{logical}}^{(\text{LDPC})}$	ϵ_{total}	ϵ	
				$p_Z = 10^{-3}$				
11	12	6.4×10^{11}	3.0×10^{12}	1.00	1.00	1.00	1.00	
	16	7.4×10^{11}	2.5×10^{12}	1.00	9.19×10^{-1}	1.00	1.00	
	22	9.0×10^{11}	2.2×10^{12}	1.00	3.89×10^{-3}	1.00	1.00	
13	12	4.7×10^{11}	3.1×10^{12}	9.62×10^{-1}	1.00	1.00	1.00	
	16	5.4×10^{11}	2.5×10^{12}	9.76×10^{-1}	9.19×10^{-1}	1.00	1.00	
	22	6.5×10^{11}	2.2×10^{12}	9.89×10^{-1}	3.89×10^{-3}	9.89×10^{-1}	9.89×10^{-1}	
15	12	3.8×10^{11}	3.2×10^{12}	6.95×10^{-2}	1.00	1.00	1.00	
	16	4.1×10^{11}	2.5×10^{12}	7.47×10^{-2}	9.19×10^{-1}	9.25×10^{-1}	9.27×10^{-1}	
	22	4.9×10^{11}	2.2×10^{12}	8.92×10^{-2}	3.89×10^{-3}	9.27×10^{-2}	1.09×10^{-1}	
17	12	3.1×10^{11}	3.4×10^{12}	1.61×10^{-3}	1.00	1.00	1.00	
	16	3.3×10^{11}	2.6×10^{12}	1.69×10^{-3}	9.24×10^{-1}	9.24×10^{-1}	9.26×10^{-1}	
	22	3.9×10^{11}	2.2×10^{12}	1.99×10^{-3}	3.89×10^{-3}	5.87×10^{-3}	2.34×10^{-2}	◀
19	12	2.6×10^{11}	3.6×10^{12}	3.64×10^{-5}	1.00	1.00	1.00	
	16	2.8×10^{11}	2.7×10^{12}	3.83×10^{-5}	9.33×10^{-1}	9.33×10^{-1}	9.34×10^{-1}	
	22	3.1×10^{11}	2.2×10^{12}	4.29×10^{-5}	3.89×10^{-3}	3.93×10^{-3}	2.15×10^{-2}	
				$p_Z = 5 \times 10^{-4}$				
11	12	6.4×10^{11}	3.0×10^{12}	9.22×10^{-1}	1.00	1.00	1.00	
	16	7.4×10^{11}	2.5×10^{12}	9.49×10^{-1}	9.79×10^{-3}	9.49×10^{-1}	9.49×10^{-1}	
	22	9.0×10^{11}	2.2×10^{12}	9.72×10^{-1}	1.90×10^{-6}	9.72×10^{-1}	9.72×10^{-1}	
13	12	4.7×10^{11}	3.1×10^{12}	2.65×10^{-2}	1.00	1.00	1.00	
	16	5.4×10^{11}	2.5×10^{12}	3.00×10^{-2}	9.79×10^{-3}	3.95×10^{-2}	4.16×10^{-2}	◀
	22	6.5×10^{11}	2.2×10^{12}	3.60×10^{-2}	1.90×10^{-6}	3.60×10^{-2}	3.81×10^{-2}	
15	12	3.8×10^{11}	3.2×10^{12}	2.98×10^{-4}	1.00	1.00	1.00	
	16	4.1×10^{11}	2.5×10^{12}	3.20×10^{-4}	9.79×10^{-3}	1.01×10^{-2}	1.22×10^{-2}	
	22	4.9×10^{11}	2.2×10^{12}	3.86×10^{-4}	1.90×10^{-6}	3.87×10^{-4}	2.54×10^{-3}	
17	12	3.1×10^{11}	3.4×10^{12}	3.35×10^{-6}	1.00	1.00	1.00	
	16	3.3×10^{11}	2.6×10^{12}	3.52×10^{-6}	1.00×10^{-2}	1.00×10^{-2}	1.22×10^{-2}	
	22	3.9×10^{11}	2.2×10^{12}	4.13×10^{-6}	1.90×10^{-6}	6.03×10^{-6}	2.16×10^{-3}	
19	12	2.6×10^{11}	3.6×10^{12}	3.81×10^{-8}	1.00	1.00	1.00	
	16	2.8×10^{11}	2.7×10^{12}	4.01×10^{-8}	1.05×10^{-2}	1.05×10^{-2}	1.26×10^{-2}	
	22	3.1×10^{11}	2.2×10^{12}	4.50×10^{-8}	1.90×10^{-6}	1.95×10^{-6}	2.15×10^{-3}	

Table 28: Effective logical volumes and logical error probabilities per shot for the factorization of an RSA-2048 number on the LDPC-cat architecture, as a function of the distances d_{rep} and d_{LDPC}, for $p_Z = 10^{-3}$ and $p_Z = 5 \times 10^{-4}$. $\text{Vol}_{\text{logical}}^{(\text{eff,rep})}$ and $\text{Vol}_{\text{logical}}^{(\text{eff,LDPC})}$ are given by Equations (144) and (146), in logical cycles of d_{rep} and d_{LDPC} physical cycles respectively, which is why $\text{Vol}_{\text{logical}}^{(\text{eff,LDPC})}$ decreases as d_{LDPC} grows. $\epsilon_{\text{logical}}^{(\text{rep})}$, $\epsilon_{\text{logical}}^{(\text{LDPC})}$ and ϵ_{total} are given by Equations (148) and (149), with Equations (122) and (147), and exclude the factories, which ϵ includes, with the factory failure probabilities of Section C.7.2. All pairs use the surgery durations of Equation (78), $T_{\text{cad}} = \max(T_{\text{surg}}^{(\text{LDPC})}, 21)$, that is 21 for $d_{\text{LDPC}} \leq 16$ and 22 for $d_{\text{LDPC}} = 22$ (Equation (123)), and $n_{\text{buf}}^{(\text{Adder})} = 3$ (Equation (124)). An entry of 1.00 denotes a probability above 0.995. The pairs marked by ◀ are selected as the pairs with the fewest physical qubits among those with $\epsilon < 10\%$ (criterion of Section C.7.2.ii): $(17, 22)$ at $p_Z = 10^{-3}$ and $(13, 16)$ at $p_Z = 5 \times 10^{-4}$.

16, 33] attribute the dominant bit flips to the CNOT gate. The probability per gate of a bit flip on either qubit is

$$p_X^{\text{CX}} = 0.5\, e^{-2\bar{n}}, \tag{151}$$

obtained by fitting numerical simulations of the gate [4, 33] and extrapolated to larger $\bar{n}$ [4]. What matters for the estimates of this appendix is therefore not just the idle bit-flip time. The gates, measurements and preparations must also preserve this bit-flip time, on every qubit of a large chip.

C.9.2 Experimental state of the art

In this section, we briefly review the measurements of the bit-flip time T_Z of dissipative cat qubits. Two-photon dissipation was first demonstrated in [20], where it confined the state of the resonator to the cat qubit manifold. The exponential suppression of bit flips was then observed in [26], up to a bit-flip time of about 1 ms, 300 times the lifetime of the undriven resonator. Above $\bar{n} \approx 5$, this bit-flip time saturated, which the authors attributed to thermal exci-

	$p_Z = 5 \times 10^{-4}$		$p_Z = 10^{-3}$	
	Repetition-cat	LDPC-cat	Repetition-cat	LDPC-cat
Distances	$d = 15$	$(13, 16)$	$d = 17$	$(17, 22)$
Physical qubits	49 649	22 593	55 497	25 673
T_{shot} (error-correction cycles)	2.493×10^{10}	2.82×10^{10}	2.598×10^{10}	3.32×10^{10}
T_{shot} (hours)	6.92	7.84	7.22	9.24
Physical volume	1.17×10^{15}	4.2×10^{14}	1.37×10^{15}	5.5×10^{14}
(qubits × error-correction cycles)		1.7×10^{14} (rep)		2.2×10^{14} (rep)
		2.4×10^{14} (LDPC)		3.4×10^{14} (LDPC)
Eff. logical volume, without factories	2.68×10^{12}	5.4×10^{11} (rep)	2.43×10^{12}	3.9×10^{11} (rep)
		2.5×10^{12} (LDPC)		2.2×10^{12} (LDPC)
Eff. logical volume, with factories	2.79×10^{12}	7.0×10^{11} (rep)	2.52×10^{12}	5.0×10^{11} (rep)
		2.5×10^{12} (LDPC)		2.2×10^{12} (LDPC)
Eff. logical qubits, without factories	1615	1681	1592	1631
		(247 rep + 1434 LDPC)		(197 rep + 1434 LDPC)
Eff. logical qubits, with factories	1680	1756	1649	1688
		(322 rep + 1434 LDPC)		(254 rep + 1434 LDPC)
Logical error/shot	0.211%	4.0%	1.25%	0.59%
Failure/shot, ϵ	0.43%	4.2%	3.0%	2.3%
Number of shots	9.24	9.60	9.48	9.42
Total runtime (days)	2.67	3.14	2.85	3.63

Table 29: Physical resources of one RSA-2048 factorization at $p_Z = 5 \times 10^{-4}$ and $p_Z = 10^{-3}$ on the repetition-cat architecture (Section C.7.2, Table 24) and on the LDPC-cat architecture (Section C.8.2), at the distances selected at each p_Z. The failure probability per shot ϵ includes the factories, and the number of shots is $9.2/(1-\epsilon)$. Physical qubits include the factories. The physical volume of one shot, in physical qubits times error-correction cycles, excludes the factories (Equations (115), (143) and (145)); on the LDPC-cat architecture, it is the sum over the two layers. The effective logical volume without factories is the one that enters the logical error probability, the factories being counted through their output infidelities; the volume with factories counts them as $n_{\text{fac}}/(2d-1)$ effective logical qubits during the whole shot, with $d = d_{\text{rep}}$ on the LDPC-cat architecture, where they belong to the repetition layer. The effective number of logical qubits is the effective logical volume divided by the number of logical cycles T_{shot}/d (Equation (118)); on the LDPC-cat architecture, it is the sum over the two layers, each counted in its own logical cycles of d_{rep} and d_{LDPC} error-correction cycles, the LDPC layer contributing $k_L = 1434$ by construction (Equation (146)).

Operation	Error	Probability
Preparation in $\|+\rangle$	Z	p_Z
X-basis measurement	outcome flipped	p_Z
CNOT	Z_c, Z_t or $Z_c Z_t$	$p_Z/3$ each
Idle, during any operation	Z	p_Z

Table 30: Generic circuit-level phase-flip noise model used throughout this work, with the single parameter p_Z (Section C.1.1). Every noisy operation is modeled as a perfect operation followed by a stochastic error, with the probability given in the third column. For the X-basis measurement, this error flips the measurement outcome. Every qubit that idles while an operation is performed undergoes a Z error with probability p_Z. Preparations in $|0\rangle$ and Z-basis measurements are insensitive to phase flips.

tations of the transmon used for the tomography. In a later device without a transmon [27], the bit flips were detected by monitoring the field radiated by the resonator. Without quantum control, the mean time between bit flips in this device reached about 100 s at $\bar{n} \approx 40$, or a bit-flip time T_Z of about 50 s. With quantum control, the bit-flip time reached 0.3 s with an autoparametric stabilization [28]. In a device with a transmon-free tomography, it reached 15 s at rest at $\bar{n} = 11.3$, and remained above 10 s under a continuous single-qubit drive [29]. Squeezed cat states reached a bit-flip time of 22 s at $\bar{n} = 4.1$ [31]. Note that all these bit-flip times were measured on a single cat qubit.

Three properties remain to be demonstrated. First, the bit-flip time must keep growing exponentially with $\bar{n}$, with quantum control, up to about 18 photons (see Section C.9.3). This exponential growth was followed up to $\bar{n} \approx 40$ only without quantum control [27], and the bit-flip time saturated in [26, 31] and, by $\bar{n} \approx 20$, in [28]. Second, all the operations, and in particular the CNOT gates, must preserve this bit-flip time. Bit-flip suppression has been observed during single-qubit Z rotations [29, 31] and, at small $\bar{n}$, during CNOT gates between a transmon and a cat qubit [36]. Since the operations add new bit-flip mechanisms, such as the one of Equation (151), they can only shorten the bit-flip time measured at rest. Last, these two properties must hold on a chip with many cat qubits. We discuss the requirement on this bit-flip time in Section C.9.3, and we present a mitigation strategy in Section C.9.4.

C.9.3 Comparison with the bit-flip assumptions of the main text

The results of the main text assume that no bit flip occurs during the computation. In practice, this means that the bit-flip error probability is negligible. In this section, we quantify what negligible means, and we estimate the bit-flip time required for each computation of this paper.

The codes of this work do not correct bit flips, and X_L has weight one on the repetition codes and on the LDPC-cat codes (Section C.1.3). Therefore, a single physical bit-flip error corresponds to a logical bit-flip error. Assuming pessimistically that any bit flip causes a logical error (some bit flips are harmless, for instance on a qubit prepared in an eigenstate of X), we count any bit flip on any physical qubit during a shot as a failure of this shot.

We denote by p_{bf} the probability of a bit flip, that is, of an X or Y error, per physical qubit and per error-correction cycle. It is related to the bit-flip time by $p_{\text{bf}} \simeq T_{\text{round}}/(2T_Z)$ (Section C.9.1), where $T_{\text{round}} = 1\,\mu\text{s}$ is the duration of an error-correction cycle. One shot involves $Q = n_{\text{total}}\, T_{\text{shot}}$ physical qubit-cycles, where n_{total} is the number of physical qubits, including the factories and the routing qubits, and T_{shot} is the duration of a shot in error-correction cycles. A shot thus fails because of bit flips with probability $1-(1-p_{\text{bf}})^Q \simeq p_{\text{bf}}\, Q$.

As for logical phase-flip errors, a shot that fails because of bit flips is detected and retried (Section A.9.0.vii). A failure probability ϵ_{bf} per shot thus multiplies the expected number of shots by $1/(1-\epsilon_{\text{bf}})$. We choose $\epsilon_{\text{bf}} = 10\,\%$, the threshold that we also impose on the per-shot failure probability due to phase flips (Section C.7.2). Indeed, increasing $\bar{n}$ suppresses bit flips at the cost of more phase flips (Equation (150)), and we expect $\bar{n}$ to be chosen in practice such that logical phase flips and logical bit flips are balanced. This budget lengthens the runtime by 11 %, and a bit-flip time ten times longer than required would bring this overhead down to about 1 %. The code distances selected in Section C.7.2 remain unchanged, since this threshold concerns only the failures due to phase flips, while bit flips multiply the number of shots by a separate factor. The budget ϵ_{bf} then gives the requirement

$$\begin{aligned} p_{\text{bf}} &\leq p_{\text{bf}}^{\max} = \frac{\epsilon_{\text{bf}}}{n_{\text{total}}\, T_{\text{shot}}}, \\ T_Z &\geq T_Z^{\min} = \frac{n_{\text{total}}\, T_{\text{shot}}\, T_{\text{round}}}{2\,\epsilon_{\text{bf}}}. \end{aligned} \tag{152}$$

This requirement concerns the bit-flip time T_Z of an idle cat qubit, and it assumes that the gates, measurements and preparations preserve this bit-flip time with T_{shot} expressed as the number of error-correction cycles per shot. As a rough guide, we translate it into a minimum mean photon number $\bar{n}_{\min} = \frac{1}{2}\ln[1/(2p_{\text{bf}}^{\max})]$ with Equation (151), assuming the equivalent of one CNOT gate per qubit and per cycle. For RSA-2048, requiring the same budget for the whole computation instead of per shot would make this requirement only nine to ten times stricter, and would increase $\bar{n}_{\min}$ by about one photon.

For RSA-2048 on the LDPC-cat architecture at $p_Z = 5\times10^{-4}$, with 22 593 physical qubits, the bit-flip probability per qubit and per cycle must remain below about 1.6×10^{-16} (see Table 31 for the other configurations). In other words, a bit flip should occur on the whole chip less than once every ten shots on average. The bit-flip time T_Z must then exceed about a hundred years, which requires about 18 photons with the rough guide above. Since $\bar{n}_{\min}$ depends only logarithmically on Q, it changes little between computations. For instance, FFDLP-512, whose shots last only a few minutes, still requires a bit-flip time of about two months and about 15 photons.

Experiments on dissipative cat qubits have already reached a noise bias of more than seven orders of magnitude, with bit-flip and phase-flip times of 15 s and 490 ns at $\bar{n} = 11.3$ [29], and of 22 s and 1.3 µs at $\bar{n} = 4.1$ [31]. The bit-flip time of about a hundred years required for RSA-2048 on the LDPC-cat architecture is still more than eight orders of magnitude longer than the bit-flip times measured with quantum control (Section C.9.2). Moreover, the bit-flip time eventually saturates in current experiments, and the mechanisms that limit it are not yet established. Candidate mechanisms include the excitation of an ancillary transmon [26, 28], the dephasing of the resonator and the thermalization of the buffer [28], and high-energy particle impacts [27, 31]. It is therefore not certain that the bit-flip time can be extended by another eight orders of magnitude. In case it cannot, we discuss and quantify a mitigation strategy in the next section.

C.9.4 Additional layer of bit-flip protection

In all the resource estimates of this paper, we assumed that bit flips are rare enough to require no active error correction, and we relied only on classical codes that correct phase flips, the repetition codes and the LDPC-cat codes [16]. As shown in Section C.9.3, this requires a bit-flip probability below about 8×10^{-17} per qubit and per cycle on the repetition-cat architecture at $p_Z = 5\times10^{-4}$, for a failure probability of 10 % per shot. In this section, we estimate the number of physical qubits required when the bit-flip probability is small but not zero. More precisely, we assume a bit-flip probability $p_X = 10^{-11}$ per fault location in the circuit-level noise model of [37, Table I], whose phase flips follow the model of Table 30.

To estimate the overhead of correcting these bit flips, we use the Elevator codes of [37], which have asymmetric distances. These codes concatenate inner repetition codes of distance d_Z, which correct phase flips, into classical outer codes that correct bit

Computation	p_Z	Architecture	Distances	n_{total}	T_{shot}	Q	$p_{\text{bf}}^{\max}$	$T_Z^{\min}$	$\bar{n}_{\min}$
RSA-2048	5×10^{-4}	Repetition-cat	$d = 15$	49 649	6.92 h	1.24×10^{15}	8.08×10^{-17}	196 years	18.2
		LDPC-cat	$(13, 16)$	22 593	7.84 h	6.38×10^{14}	1.57×10^{-16}	101 years	17.8
RSA-2048	10^{-3}	Repetition-cat	$d = 17$	55 497	7.22 h	1.44×10^{15}	6.93×10^{-17}	228 years	18.3
		LDPC-cat	$(17, 22)$	25 673	9.24 h	8.54×10^{14}	1.17×10^{-16}	135 years	18.0
ECDLP-256	5×10^{-4}	LDPC-cat	$(13, 16)$	29 588	60 min	1.07×10^{14}	9.39×10^{-16}	16.9 years	17.0
FFDLP-512	5×10^{-4}	Repetition-cat	$d = 11$	9 336	2.28 min	1.28×10^{12}	7.84×10^{-14}	2.4 months	14.7
		LDPC-cat	$(11, 12)$	6 656	2.51 min	1.00×10^{12}	9.98×10^{-14}	1.9 months	14.6

Table 31: Requirement on bit flips of Equation (152), for a budget $\epsilon_{\text{bf}} = 10\,\%$ per shot, assuming that any bit flip on any physical qubit makes the shot fail. The configurations are those selected in Sections C.7.2, C.8.2, D.2 and D.3, with the distance d on the repetition-cat architecture and the distances $(d_{\text{rep}}, d_{\text{LDPC}})$ on the LDPC-cat architecture. n_{total} is the number of physical qubits, including the factories and the routing qubits, and T_{shot} is the duration of one shot, rounded up to 60 min for ECDLP-256. $Q = n_{\text{total}}\, T_{\text{shot}}$ is the number of physical qubit-cycles of one shot, with T_{shot} in error-correction cycles of $T_{\text{round}} = 1\,\mu\text{s}$. $p_{\text{bf}}^{\max}$ is the maximum bit-flip probability per qubit and per cycle, and $T_Z^{\min}$ is the minimum bit-flip time of an idle qubit. $\bar{n}_{\min}$ is the minimum mean photon number given by Equation (151) with the equivalent of one CNOT gate per qubit and per cycle.

flips, using native transversal operations. We use the $[15, 9, 3]$ outer code, in which 15 inner repetition codes encode 9 logical qubits, with $d_X = 3$ and d_Z the distance of the chosen inner repetition codes. The $[15, 9, 3]$ Elevator code has been simulated as a quantum memory to obtain a logical error model. We take the phase-flip model from [37]. The bit-flip model comes from new memory simulations performed with the Tesseract decoder [114]. These simulations use the same noise model as [37], with $d_Z = 9$, 11, 13 and 15, 10^6 shots per point and $10^{-6} < p_X < 5 \times 10^{-5}$. The resulting logical error models, per logical qubit and per d_Z rounds of syndrome extraction of the repetition codes, are

$$\begin{aligned} p_{Z_L}(p_Z, d_Z) &= 0.12\, d_Z\, (34.4\, p_Z)^{0.94\,(d_Z+1)/2}, \\ p_{X_L}(p_X, d_Z) &= d_Z^{4.10}\, (17.78\, p_X)^{2.06}. \end{aligned} \tag{153}$$

These simulations only concern a quantum memory, since no logical gate set has been designed for Elevator codes yet. We therefore assume a logical cycle of d_Z rounds of syndrome extraction of the repetition codes. This assumption is conservative, since more efficient schemes, for instance based on transversal gates, may yet be developed. Under this assumption, we estimate the number of physical qubits by converting the effective logical volume of the repetition-cat estimate (Section C.7.2) to the Elevator code model.

At $p_Z = 5 \times 10^{-4}$, the repetition-cat architecture has, including the factories, an effective logical volume $\text{Vol}_{\text{logical}}^{(\text{eff})} = 2.79 \times 10^{12}$ and $n_{\text{logical}}^{(\text{eff})} = 1680$ effective logical qubits (Table 29). We choose $d_Z = 17$. With Equations (117) and (153), and $p_X = 10^{-11}$, we obtain the per-shot logical error probabilities $\epsilon_{\text{logical}}^{(Z)} = 6.7 \times 10^{-3}$ and $\epsilon_{\text{logical}}^{(X)} = 2.5 \times 10^{-3}$, due to phase flips and to bit flips, respectively. The probability of no logical error during a shot is thus above 99 %. The total qubit overhead of the $[15, 9, 3]$ Elevator code is

$$n_{\text{over}} = \frac{16}{9}\,(2d_Z - 1). \tag{154}$$

Using this and $n_{\text{logical}}^{(\text{eff})} = 1680$, we find that the overall number of physical qubits required is 98 560.

At $p_Z = 10^{-3}$, the repetition-cat architecture has, including the factories, $\text{Vol}_{\text{logical}}^{(\text{eff})} = 2.52 \times 10^{12}$ and $n_{\text{logical}}^{(\text{eff})} = 1649$. If a $d_Z = 21$ repetition code is used, the per-shot logical error probabilities are $\epsilon_{\text{logical}}^{(Z)} = 4.7 \times 10^{-3}$ and $\epsilon_{\text{logical}}^{(X)} = 5.5 \times 10^{-3}$, and the probability of no logical error during a shot is above 98 %. With Equation (154), the overall number of physical qubits required is 120 194.

Using LDPC-cat codes [16] as inner codes should further reduce this overhead [115]. Until this construction is available, we estimate the overhead of such a concatenated LDPC-cat code by assuming the same proportional increase as from the repetition-cat architecture to the Elevator code. At $p_Z = 5 \times 10^{-4}$, this increase is 99 %, and the number of physical qubits of the LDPC-cat architecture goes from 22 593 to 44 850. At $p_Z = 10^{-3}$, the increase is 117 %, and the number of physical qubits goes from 25 673 to 55 602.

Conversely, with the budget $\epsilon_{\text{logical}}^{(X)} = \epsilon_{\text{bf}} = 10\,\%$ of Section C.9.3, the Elevator code tolerates a bit-flip probability of order 10^{-11} per fault location. Since a repetition code holds on average about two fault locations per qubit per cycle, this corresponds to a bit-flip time of about one hour, which requires about 11 photons with the rough guide of Section C.9.3. Without an outer code, the same budget requires a bit-flip time of about 200 years and about 18 photons (Table 31). Table 32 summarizes these numbers for both values of p_Z, together with a rough estimate for the LDPC-cat architecture. The outer code thus relaxes the requirement on the bit-flip time by a factor of more than 10^6, or about seven photons, at the cost of about twice as many physical qubits.

These estimates rest on assumptions that remain to be checked. First, the bit-flip model, fitted for

Computation	p_Z	Architecture	Distances	n_{total}	Overhead	$T_Z^{\min}$ without	$T_Z^{\min}$ with	Reduction	$\bar{n}_{\min}$
RSA-2048	5×10^{-4}	Repetition-cat	$d_Z = 17$	98 560	+99% (from 49 649)	196 years	1.2 hours	1.5×10^6	11.1
		LDPC-cat	estimate	44 850	+99% (from 22 593)	101 years	1.3 hours	7.1×10^5	11.1
RSA-2048	10^{-3}	Repetition-cat	$d_Z = 21$	120 194	+117% (from 55 497)	228 years	1.7 hours	1.2×10^6	11.3
		LDPC-cat	estimate	55 602	+117% (from 25 673)	135 years	1.8 hours	6.8×10^5	11.3

Table 32: Physical qubit count and requirement on bit flips with the additional Elevator layer, to be compared with Table 31, for a logical bit-flip failure probability $\epsilon_{\text{logical}}^{(X)} = 10\,\%$ per shot. n_{total} is the number of physical qubits with the outer code, and the overhead is the increase of n_{total} relative to the number of physical qubits without it. $T_Z^{\min}$ is the minimum bit-flip time of an idle qubit, without the outer code (from Table 31) and with it, and the reduction is the ratio of these two values of $T_Z^{\min}$. $\bar{n}_{\min}$ is obtained as in Table 31. The LDPC-cat rows are rough estimates: the number of physical qubits assumes the same proportional increase as the repetition-cat rows, and $T_Z^{\min}$ applies the logical bit-flip model of Equation (153), per logical qubit and with the same d_Z, to the effective logical volume of the LDPC-cat architecture including the factories (Table 29).

$p_X > 10^{-6}$ and $d_Z \leq 15$, is used at $p_X = 10^{-11}$, five orders of magnitude lower, and at $d_Z = 17$ and 21. Similarly, the phase-flip model is used at values of p_Z five to ten times lower and at larger distances than in the simulations of [37]. Second, we assume that the factories, the routing qubits and the logical operations are protected against bit flips at the logical rate of the memory, although their Elevator constructions remain to be designed.

D Estimation for the other variations and space-time trade-offs

D.1 RSA

D.1.1 Space-efficient configuration

Reference [47] proposes three layouts of the two-level $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory, and Appendix C uses the third, which is the fastest. The space-efficient configuration replaces the third layout with the second [47, Table III], which takes 560 instead of 1566 physical qubits and a length of 8 instead of 29 columns. The second layout produces one state every 61.5 instead of 20.75 cycles, hence $T_{\mathrm{prod}}^{|\mathrm{C}_X\mathrm{C}_X X\rangle} = 62$. Its output infidelity is slightly lower, 1.81×10^{-13} instead of 2.02×10^{-13} at $p_Z = 5 \times 10^{-4}$. Everything else is as in the balanced configuration of Section C.8.2: the LDPC-cat architecture, $p_Z = 5 \times 10^{-4}$, the algorithmic parameters of Table 22, the $|i\rangle$ factories and the distance selection criterion of Section C.7.2.ii.

Equation (123) gives $T_{\mathrm{cad}} = 62$ at every pair of distances of Table 28, and Equation (82) still gives one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory and two $|i\rangle$ factories. Adders last $2nT_{\mathrm{cad}}$ (Equation (135)) and the And gates of Phaseups are issued at the cadence (Equation (137)), so that both durations grow with T_{cad}. In a Lookup, the condition $T_{\mathrm{prod}}^{|\mathrm{C}_X\mathrm{C}_X X\rangle} \le 2T_{\mathrm{surg}}$ of Section C.6 no longer holds, so that the $2^{w-1}-1$ And gates performed together with a parity-Cnot gate also wait for the factory. The duration of a Lookup then takes the general form of [47, Eq. (4.7)]

$$T_{\textsc{Lookup}}(w) = (2^{w-1} - w)\, T_{\mathrm{cad}} + T_{\mathrm{surg}} + (2^{w-1} - 1)\max(2T_{\mathrm{surg}}, T_{\mathrm{cad}}), \tag{155}$$

with $T_{\mathrm{surg}} = d$ on the repetition-cat architecture and $T_{\mathrm{surg}} = T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ on the LDPC-cat architecture. For $T_{\mathrm{cad}} \le 2T_{\mathrm{surg}}$, Equation (155) coincides with Equations (109) and (136). Here, $T_{\mathrm{cad}} > 2T_{\mathrm{surg}}^{(\mathrm{LDPC})}$ at every pair, and Equation (155) reduces to $(2^w - w - 1)\, T_{\mathrm{cad}} + T_{\mathrm{surg}}^{(\mathrm{LDPC})}$. A Lookup(6) then lasts 3550 cycles instead of 1554, and a Lookup(5) 1628 instead of 727.

At $T_{\mathrm{cad}} = 62$, Equation (124) would give a single Adder buffer. The buffer zone, however, also holds the auxiliary qubits of Lookups and the workspace of Phaseups, which use up to 20 of its $9n_{\mathrm{buf}}^{(\textsc{Adder})}$ positions (Table 26), hence $n_{\mathrm{buf}}^{(\textsc{Adder})} \ge 3$. The buffers are therefore kept as in the balanced configuration, $n_{\mathrm{buf}}^{(\textsc{Adder})} = 3$ and $n_{\mathrm{buf}}^{(\textsc{Lookup})} = 2$. Adders then hold three buffers where one suffices, a slight overestimate of their volume and of their $|i\rangle$ count. The factory zone has length $l_{\mathrm{fac}} = 8 + 2 \times 10.5 = 29$ and holds $n_{\mathrm{fac}} = 560 + 2 \times 153 = 866$ physical qubits, so that the factory routing zone of Equation (128) holds $2h_{\mathrm{routing,fac}}\, l_{\mathrm{fac}} - (l_{\mathrm{fac}} - l_{\mathrm{buf}}) = 578$ physical qubits instead of 977. The factories and their routing thus take 1444 instead of 2849 physical qubits.

Table 33 compares the two configurations. The criterion again selects $(d_{\mathrm{rep}}, d_{\mathrm{LDPC}}) = (13, 16)$, with 21 188 physical qubits. One shot lasts 6.89×10^{10} cycles, about 19.15 h. The longer shot raises the logical error probabilities to $\epsilon_{\mathrm{logical}}^{(\mathrm{rep})} = 6.7 \times 10^{-2}$ and $\epsilon_{\mathrm{logical}}^{(\mathrm{LDPC})} = 2.4 \times 10^{-2}$. The $|\mathrm{C}_X\mathrm{C}_X X\rangle$ and $|i\rangle$ factories add 1.4×10^{-4} and 2.0×10^{-3}, hence $\epsilon = 9.1\%$, $9.2/(1-\epsilon) = 10.13$ shots and a total runtime of 8.08 d. Compared with the balanced configuration, the physical qubit count decreases by 6.2% and the total runtime grows by a factor of 2.58. The factories and their routing hold only 13% of the physical qubits of the balanced configuration, whereas the cadence they set governs most of the duration of a shot.

D.1.2 Time-efficient configuration

The estimate of Section C.7.2 runs one shot after the other on one chip. Two levers shorten the clock time at the price of more qubits. The first runs several shots at once on several identical chips. The second runs several residue engines at once on one chip, sharing a single exponent register, as proposed in [17, Sec. VI.B.1]. We keep every algorithmic parameter fixed (s, κ, S, the window sizes and the prime set), as well as $p_Z = 5 \times 10^{-4}$, and we treat RSA-2048 on the repetition-cat chip, then on the LDPC-cat chip.

i Parallel shots. Shots are independent once g is fixed, so k identical chips can run them in rounds of k simultaneous shots, on $k\, n_{\mathrm{total}}$ physical qubits. The expected clock time is

$$T_{\mathrm{wall}} = T_{\mathrm{shot}}\, \mathbb{E}[N_{\mathrm{rounds}}(k)]\,, \tag{156}$$

where $N_{\mathrm{rounds}}(k)$ is the number of rounds needed to collect $s+1$ consistent shots that pass the lattice step. Each shot keeps the statistics of the serial run (Section A.9), with a fresh g after a failed lattice step. For $k = 1$, N_{rounds} is the number of shots R, with mean $\mathbb{E}[R]$ given by Equation (60) divided by $1 - \epsilon$.

ii Parallel residue engines. Three properties of the oracle make the second lever cheap. First, $\tilde{V}$ is a sum of contributions, one per prime of the RNS, and each contribution only needs read access to the exponent register. Second, the exponent register is only used as a Lookup address and as a control, and all phase corrections on it are deferred and merged into one final Phaseup per window. Third, almost all of a shot is spent inside the loop over the primes (Table 25), while the exponent register holds most of the qubits. On the repetition-cat chip, at $d = 15$, the exponent register takes 38 400 of the 49 649 physical qubits, counting one ancilla per data qubit as in Table 23.

	$\lvert C_X C_X X\rangle$			Physical		T_{shot}				Runtime	
Configuration	layout	T_{cad}	$(d_{\text{rep}}, d_{\text{LDPC}})$	qubits	Factories	cycles	hours	ϵ	Shots	(days)	Qubit-days
Balanced	third	21	$(13, 16)$	22 593	2849	2.82×10^{10}	7.84	4.2%	9.60	3.14	7.09×10^4
Space-efficient	second	62	$(13, 16)$	21 188	1444	6.89×10^{10}	19.15	9.1%	10.13	8.08	1.71×10^5

Table 33: RSA-2048 on the LDPC-cat architecture at $p_Z = 5 \times 10^{-4}$, with the third (balanced, Section C.8.2) and the second (space-efficient) layouts of the $\lvert C_X C_X X\rangle$ factory [47, Table III]. The Factories column includes the factory routing zone. The number of shots is $9.2/(1-\epsilon)$, and the qubit-days are the physical qubit count times the total runtime.

We follow the construction of [17, Sec. VI.B.1]. The chip holds ρ working registers, each with its own residue registers, workspace and local accumulator, and the $|P|$ primes are processed in $B = \lceil |P|/\rho \rceil$ batches. Batch b holds $m_b = \min(\rho, |P| - (b-1)\rho)$ primes. In each batch, every register runs the subroutines of Figure 26 on its own prime. Register $\#0$ accumulates its contribution into $|\sigma\rangle$, and register $\#j \geq 1$ into a local accumulator L_j initialized to $|0\rangle$. The local accumulators are then added into $|\sigma\rangle$ one after the other, and each L_j is returned to $|0\rangle$ by uncomputing its accumulation while its residue is still available. Figure 54 shows the circuit for $\rho = 2$.

Three details of Figure 26 change. Register $\#j$ handles the primes $p_j, p_{j+\rho}, p_{j+2\rho}, \ldots$, so the merged uncomputation of ComputeExponent uses the differences between consecutive primes of that sequence. The sums still telescope, so each register starts from $|0\rangle$ and, after the loop, uncomputes its own last exponent. The deferred phase corrections of all registers act on the same exponent windows and are diagonal. These corrections therefore commute with the operations in between, and the single final Phaseup applies them together, merging their phase tables. The registers read the exponent windows in the same order and at the same time, so that each access to an exponent qubit serves all registers at once. Operations controlled by the exponent register that act on different registers commute, so the registers can share the exponent register without copying it. Unlike in [17], no Clifford frame cleaning is needed, since the lattice surgery circuits of Appendix C execute their Clifford gates physically.

iii Modular merge. Both $|\sigma\rangle$ and the local accumulators L_j hold values reduced modulo $M = N \gg \bar{\kappa}$ (Equation (51)). The merge must therefore add modulo M,

$$|\sigma\rangle\,|L_j\rangle \longmapsto |(\sigma + L_j) \bmod M\rangle\,|L_j\rangle\,, \tag{157}$$

and an ordinary $(\kappa+1)$-qubit Q-Q Adder does not suffice. We use four additions of $\kappa+1$ bits. The first computes $s = \sigma + L_j$. The second subtracts M with a Q-C Adder and copies the sign bit to a no-wrap flag $f = [s < M]$. This subtraction is safe in $\kappa+1$ bits, since $-M \leq s - M \leq M - 2$ and $M \leq 2^\kappa$. The third adds fM, prepared in the operand slots by a fan-out of f, which leaves $(\sigma + L_j) \bmod M$ in the accumulator. The fan-out is then reversed, which returns the operand slots to $|0\rangle$ before f is cleared. The fourth, a Q-Q Comparison(κ), adds $[\sigma \geq L_j]$ to f, where σ now holds the reduced sum. Without a wrap, this sum is $s \geq L_j$. With a wrap, it is $s - M < L_j$, since the original σ is below M. Hence $[\sigma \geq L_j] = f$, including at the boundary $s = M$, and the Comparison returns f to $|0\rangle$. The three parity-Cnot surgeries, which copy the sign bit, fan out f and reverse the fan-out, account for the term $3d$ below. L_j makes four register moves. L_j enters the compute zone of register $\#0$ for the first step and returns to its engine during the second and third steps, which need the operand slots for M and fM. L_j then comes back for the Comparison and returns to its engine for its erasure. The flag f uses the underflow qubit of AccumulateResidue. We found no capacity conflict in that compute zone, but a detailed spatial allocation of the merge remains to be done. With the durations of Equation (108) and register moves of three qubits per d cycles (Section C.7.1), the merge lasts

$$T_{\text{merge}}(d) = 4\,T_{\text{Adder}}(\kappa+1) + 3d + 4\,\lceil \kappa/3 \rceil\, d, \tag{158}$$

that is 6417 cycles at $d = 15$, and consumes 132 Toffoli gates and 540 $|i\rangle$ states.

The local accumulators are merged into $|\sigma\rangle$ one after the other, never into each other. Each L_j therefore holds exactly its own contribution until it is erased, and the inverse of its AccumulateResidue returns it to $|0\rangle$. A reduction tree would shorten the merge, but it turns some local accumulators into sums of several contributions, and these partial sums must be uncomputed before the erasures, as accounted for in [17, Eq. (26)]. For $\rho \leq 7$, the sequential merge takes at most six merges per batch, against up to five merge layers for a tree with its cleanup, so we keep the simpler sequential merge. Each erasure starts as soon as its own merge ends and overlaps the later merges.

iv Layout on the repetition-cat chip. Each extra engine brings its own buffer, compute and memory zones, $l_{\text{eng}} = l_{\text{buf}} + l_{\text{comp}} + l_{\text{mem}} = 181$ repetition-code columns, its own factories and the routing above its zones. The exponent register is shared. We keep the two-rectangle layout of Section C.7.1 and place the ρ engines in the bottom rectangle. The chip length

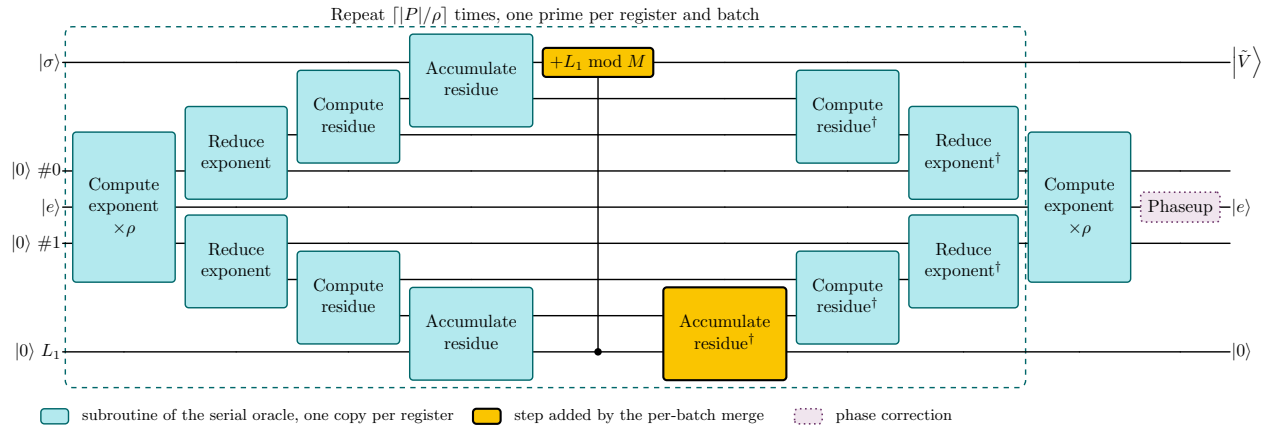


Figure 54: The oracle of Figure 26 with $\rho = 2$ working registers, following the per-batch merge of [17, Sec. VI.B.1]. Register #0 is drawn above $|e\rangle$ and register #1 mirrored below it. Each register runs the subroutines of Figure 26 on its own primes $p_j, p_{j+\rho}, p_{j+2\rho}, \ldots$, and both read the same windows of $|e\rangle$ at the same time. In each batch, register #0 accumulates into $|\sigma\rangle$ and register #1 into a local accumulator L_1 initialized to $|0\rangle$. L_1 is then added into $|\sigma\rangle$ modulo M, Equation (157), and returned to $|0\rangle$ by uncomputing its accumulation while the residue is still available. For larger ρ, the local accumulators $L_1, \ldots, L_{\rho-1}$ are added into $|\sigma\rangle$ one after the other, and each is erased right after its own merge. After the loop, each register uncomputes its last exponent, and the deferred phase corrections of all registers are applied by the single final PHASEUP.

and the bottom exponent zone become

$$l_{\mathrm{tot}}(\rho) = \left\lceil \frac{n_e + \rho\, l_{\mathrm{eng}}}{2} \right\rceil, \qquad l_{\mathrm{exp},1}(\rho) = l_{\mathrm{tot}}(\rho) - \rho\, l_{\mathrm{eng}}. \tag{159}$$

The layout requires $l_{\mathrm{exp},1} \geq 0$, that is $\rho\, l_{\mathrm{eng}} \leq n_e + 1$ for integer lengths, which allows up to $\rho = 7$ engines for RSA-2048. Larger ρ needs engines in both rectangles and is not considered here.

The routing zone between the two exponent zones gives the engines access to the exponent qubits. Its $h_{\mathrm{exp}} = 2$ rows suffice for one engine because the accesses are sequential. With ρ engines, these rows form a shared lane, in which the engines read the same exponent qubit at the same time through a tree-shaped repetition code that branches after the routing zone between the two exponent zones. With the counting convention of Equations (104) to (106), the qubit count becomes

$$\begin{aligned} n_{\mathrm{total}}(\rho, d) &= (4d-1)\, l_{\mathrm{tot}}(\rho) + 2h_{\mathrm{exp}}\, l_{\mathrm{exp},1}(\rho) \\ &\quad + \rho\big[2h_{\mathrm{routing}}\, l_{\mathrm{eng}} + 2h_{\mathrm{routing,fac}}\, l_{\mathrm{fac}} + n_{\mathrm{fac}}\big], \end{aligned} \tag{160}$$

with $h_{\mathrm{routing}} = 4$, and the global routing of Equation (105) becomes $\Gamma_{\mathrm{global}}(\rho, d) = 2d\rho + 2h_{\mathrm{exp}}\, l_{\mathrm{exp},1}(\rho) + \rho\,(2h_{\mathrm{routing}}\, l_{\mathrm{eng}} + 2h_{\mathrm{routing,fac}}\, l_{\mathrm{fac}})$. Both reduce to Equations (105) and (106) at $\rho = 1$ for every d.

The runtimes below assume that the engines never stall each other on their way to the exponent qubits or to $|\sigma\rangle$. The engines run in lockstep when they read the exponent register, so that the two rows carry one access at a time, as for a single engine. The merged tree measures the X-parity of the exponent qubit with each engine, since the exponent register is only accessed through X_L.

v Duration. We split the serial shot of Equation (110) into the loop over the primes and two pieces of fixed work. As in Section C.7, this shot excludes the final QFT. The first piece is the final uncomputation of the exponent, $T_{\mathrm{end}}(d) = \lceil n_e/w_1 \rceil \big(T_{\textsc{Lookup}}(w_1, d) + T_{\textsc{Adder}}(\ell + \log n_e) + 1\big)$, that is 597 916 cycles at $d = 15$. The second is the final PHASEUP, $T_{\mathrm{last}}(d)$. The serial time per prime is $t_{\mathrm{loop}}(d) = \big(T_{\mathrm{shot}}(d) - T_{\mathrm{end}}(d) - T_{\mathrm{last}}(d)\big)/|P|$. With ρ engines, the final uncomputations run in parallel, so T_{end} is paid once. A batch of $m_b > 1$ primes adds $m_b - 1$ merges and the erasure of the last local accumulator. One erasure costs one ACCUMULATERESIDUE, $t_{\mathrm{acc}}(d)$, including its slices of T_{other}, that is 19 014 cycles at $d = 15$. Hence

$$\begin{aligned} T_{\mathrm{shot}}(\rho, d) &= B\, t_{\mathrm{loop}}(d) + T_{\mathrm{end}}(d) + T_{\mathrm{last}}(d) \\ &\quad + \sum_{b=1}^{B} \mathbf{1}_{m_b > 1}\big[(m_b - 1)\, T_{\mathrm{merge}}(d) + t_{\mathrm{acc}}(d)\big]. \end{aligned} \tag{161}$$

Relative to an ideal division of the loop by ρ, the merges and erasures add 2.3 % to the shot at $\rho = 2$ and 5.2 % at $\rho = 7$. We do not overlap the merges with the next batch.

vi Volume and factory errors. We evaluate the effective logical volume of Equation (116) on the parallel schedule. During the loop, the ρ registers run the same sequence of operations, so each slice s of Table 23 holds $n_e + \rho\,(n_{\mathrm{col}}(s) - n_e)$ repetition-code columns, the shared exponent register being counted once. The slices of T_{other}, as in Section C.7, and the merges and erasures are counted at the peak

$$n_{\mathrm{peak}}(\rho) = n_e + \rho\,(1460 - n_e) = 1280 + 180\,\rho, \tag{162}$$

an upper bound, since 1460 columns include the exponent register of the serial run. The final uncom-

putation holds the columns of ComputeExponent in every register. The global routing and the exposed tops follow the layout of Equation (160), with $n_{e,2} = n_e - l_{\mathrm{exp},1}(\rho)$.

The merges, the erasures and the extra final uncomputations consume additional factory outputs. With $|P| - B$ local contributions,

$$\begin{aligned} N_T(\rho) &= N_T + (|P| - B)\big(N_T^{\mathrm{acc}} + N_T^{\mathrm{merge}}\big) \\ &\quad + (\rho - 1)\lceil n_e/w_1 \rceil N_T^{\mathrm{exp}}, \\ \epsilon_{\mathrm{fac}}(\rho) &= 1 - (1 - p_T)^{N_T(\rho)}\,(1 - p_i)^{N_i(\rho)}, \end{aligned} \tag{163}$$

and the same expression for $N_i(\rho)$ with the $|i\rangle$ counts. Here, N_T^{acc}, N_T^{merge} and N_T^{exp} count the Toffoli gates of one AccumulateResidue, one merge and one window of ComputeExponent. The output infidelities p_T and p_i are those of Table 19. At $\rho = 7$, the Toffoli count grows by 1.64 % and the $|i\rangle$ count by 2.07 %, which raises the factory failure probability from 0.219 % to 0.224 % per shot.

The per-shot failure probability combines the logical errors of the data and routing qubits, Equation (117), with $\epsilon_{\mathrm{fac}}(\rho)$.

vii Scope of the estimate. We base our estimate on the configurations with $d = 15$, because they stay closest to the analysis of Section C.7.2. The routing zone between the two exponent zones is kept as in that section, so that only the engine zones, their factories and their routing are replicated. The qubit count then follows directly from Equations (104) to (106). Keeping $d = 15$ for every ρ satisfies the criterion of Section C.7.2, since ϵ then stays below 0.5 %, well under its 10 % threshold. The full criterion also minimizes the number of qubits, whereas we deliberately keep $d = 15$ fixed. From $\rho = 2$ on, $d = 13$ also meets the threshold and saves 10 to 11 % of the qubits. For every configuration, including those at $d = 15$, the runtimes also rest on two scheduling assumptions that Appendix C does not cover. The first is that the engines never stall each other on the exponent qubits or on $|\sigma\rangle$. The second is that the merges do not overlap the next batch. A proper evaluation of these configurations requires the same layout, schedule and volume analysis as in Section C.7, which we leave to future work.

viii Results. Figure 55 and Table 34 give the results, with our estimate at $d = 15$ in the top rows. Three observations stand out. First, the speed-up is close to ρ. $\rho = 3$ costs 37 % more qubits than the serial run (68 244 against 49 649) and shortens the expected total runtime by a factor 2.9, from 64.0 h to 21.9 h. At the layout limit, $\rho = 7$ costs 112 % more qubits and is 6.7 times faster, 9.6 h in total. Second, the effective logical volume falls with ρ, to 0.44 times the serial one at $\rho = 3$ and 0.28 times at $\rho = 7$, but more and more slowly. Parallelism reduces the idle volume of the exponent register but not the volume of the engines themselves. Third, the two levers combine. On 10 chips with $\rho = 7$, the expected clock time is 1.06 h on 1 054 340 physical qubits.

Since the volume falls, the per-shot failure probability at $d = 15$ drops from 0.43 % at $\rho = 1$ to 0.28 % at $\rho = 7$, and $d = 13$ meets the threshold from $\rho = 2$ on. At $\rho = 2$, the margin is thin, with $\epsilon = 9.8$ %. These configurations, in the bottom rows of Table 34, save 10 to 11 % of the qubits but do not shorten the runtime, because the higher ϵ calls for more shots.

The last column of Table 34 exceeds the time per shot because of the extra rounds of Equation (156). At $d = 15$ the excess is small, with 1.02 rounds on average at $\rho = 1$ and 1.02 as well at $\rho = 3$. It grows at $d = 13$, where ϵ reaches several percent, with 1.23 rounds at $\rho = 3$. On parallel chips, the configurations at $d = 15$ are therefore also faster. At $\rho = 3$ on 10 chips, $d = 15$ gives 2.42 h against 2.79 h at $d = 13$, for 68 244 instead of 60 948 qubits per chip. All these numbers rest on the scheduling assumptions of Section D.1.2.vii and on the heuristic shot model of Equation (156).

ix LDPC-cat chip. The construction carries over to the LDPC-cat chip of Section C.8. We keep the pair $(d_{\mathrm{rep}}, d_{\mathrm{LDPC}}) = (13, 16)$ selected in Section C.8.2 for every ρ, as we keep $d = 15$ on the repetition-cat chip. The exponent register again holds most of the chip. Its n_e logical qubits take about 16 000 of the 22 593 physical qubits (71 %), counting the LDPC layer and the access zone below it. Each extra engine appends its work zone, $l_{\mathrm{work}} = l_{\mathrm{mem}} + l_{\mathrm{comp}}$ columns, to the LDPC block, so that

$$\begin{aligned} L(\rho) &= l_{\mathrm{exp}} + \rho\, l_{\mathrm{work}}, \qquad k_L(\rho) = 2L(\rho), \\ \Gamma_{\mathrm{LDPC}}(\rho) &= 2(H - 1)\, L(\rho) + 2\Delta_{\mathrm{LDPC}}. \end{aligned} \tag{164}$$

In the repetition layer, each engine brings the access zone and the proxy row under its work zone, its buffer routing zone of length $l_{\mathrm{work}} + 2 + 9(n_{\mathrm{buf}}^{(\textsc{Adder})} - 1)$, its $n_{\mathrm{buf}}^{(\textsc{Adder})}$ Adder buffers, and its factories with their routing. The access zone under the exponent zone, the left staircase and the right end of the block are shared. Equation (129) becomes

$$\begin{aligned} n_{\mathrm{total}}^{(\mathrm{LDPC})}(\rho) &= \Gamma_{\mathrm{LDPC}}(\rho) + (2H - 1)\big(L(\rho) + p_r\big) \\ &\quad + 2\Delta_{\mathrm{L}} + 2h_{\mathrm{routing,buf}}\, p_r + \rho\big[2(l_{\mathrm{work}} + 2) \\ &\qquad + 2h_{\mathrm{routing,buf}}\big(l_{\mathrm{work}} + 2 + 9(n_{\mathrm{buf}}^{(\textsc{Adder})} - 1)\big) \\ &\qquad + 18\, n_{\mathrm{buf}}^{(\textsc{Adder})}\, d_{\mathrm{rep}} + 2h_{\mathrm{routing,fac}}\, l_{\mathrm{fac}} \\ &\qquad - \max(l_{\mathrm{fac}} - l_{\mathrm{buf}}, 0) + n_{\mathrm{fac}}\big], \end{aligned} \tag{165}$$

which reduces to Equation (129) at $\rho = 1$ and adds 6410 physical qubits per engine. The block only grows

			Physical qubits			Expected total runtime	
ρ	d	ϵ	per chip	Time per shot	$\mathbb{E}[R]$	1 chip	10 chips
Estimate, $d = 15$							
1	15	0.43 %	49.6 k	6.92 h	9.25	64.0 h	7.07 h
2	15	0.34 %	58.9 k	3.54 h	9.24	32.7 h	3.62 h
3	15	0.32 %	68.2 k	2.38 h	9.24	21.9 h	2.42 h
5	15	0.29 %	86.8 k	1.44 h	9.23	13.3 h	1.47 h
7	15	0.28 %	105.4 k	1.04 h	9.23	9.6 h	1.06 h
Alternative, $d = 13$							
2	13	9.75 %	52.3 k	3.39 h	10.20	34.6 h	4.45 h
3	13	7.51 %	60.9 k	2.28 h	9.95	22.7 h	2.79 h
5	13	5.69 %	78.1 k	1.38 h	9.76	13.5 h	1.60 h
7	13	4.92 %	95.2 k	1.00 h	9.68	9.7 h	1.13 h

Table 34: RSA-2048 on the repetition-cat chip with ρ residue engines sharing one exponent register, following [17, Sec. VI.B.1], with the sequential modular merge of Section D.1.2.iii, at $p_Z = 5 \times 10^{-4}$. The top rows are our estimate, at the distance $d = 15$ of Section C.7.2 for every ρ. The bottom rows are at $d = 13$, shown only where $d = 13$ also meets the criterion $\epsilon < 10\,\%$. Qubits are given by Equation (160). The times assume that the engines never stall each other. $\mathbb{E}[R]$ and the total runtimes follow the shot model of Equation (156), and the last column runs the shots in rounds of 10 simultaneous shots on 10 identical chips. Classical pre- and post-processing are not included.

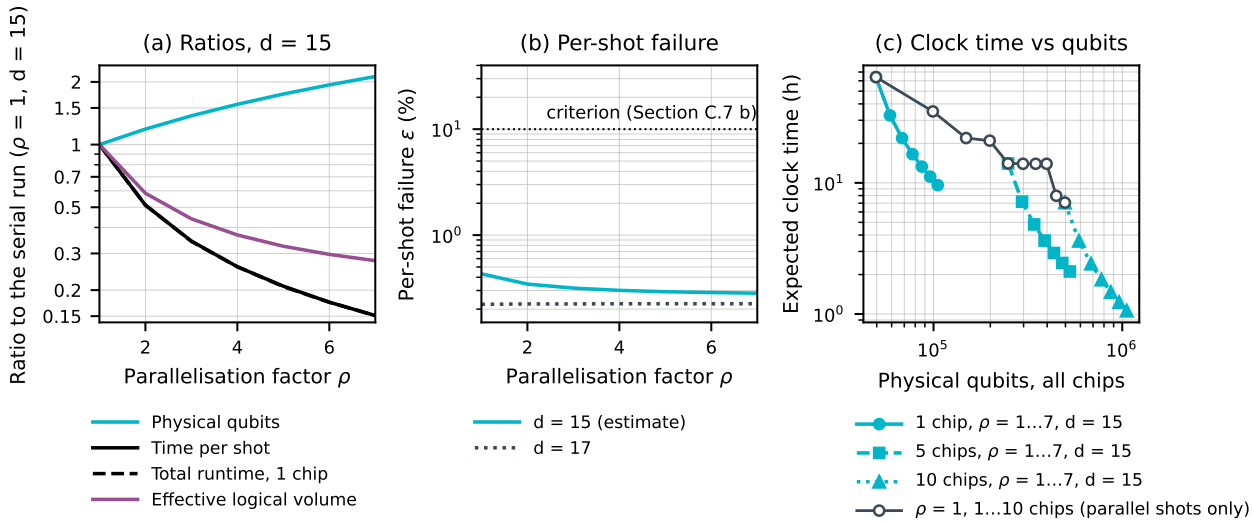


Figure 55: Space-time trade-off for RSA-2048 on the repetition-cat chip at $p_Z = 5 \times 10^{-4}$, with $\rho \leq 7$ residue engines sharing one exponent register after Ref. [17, Sec. VI.B.1], the limit of the layout of Equation (158). (a) Ratios to the serial run at $d = 15$, with the qubit count of Equation (159). The effective logical volume is that of Equation (115) on the parallel schedule. (b) Per-shot failure probability at $d = 15$, our estimate, and at $d = 17$. (c) Expected clock time against the total number of physical qubits at $d = 15$, for one, five and ten chips, compared with parallel shots alone, with the shot model of Equation (155). The runtimes assume no routing stalls.

longer, so the layout does not limit ρ, but we stop at $\rho = 7$ for comparison with the repetition-cat chip.

The address-routing rows, of $\Gamma_{\text{addr}} = 4l_{\text{exp}}$ physical qubits, are shared. The engines read the exponent tips through the same two rows of the access zone, which Section C.8.1 sizes for sequential accesses. As with the shared lane of the repetition-cat chip, the engines read the same exponent tip at the same time through a tree-shaped repetition code that branches after these rows.

The modular merge keeps its four additions. Its three parity-CNOT gates last $T_{\text{surg}}^{(\text{LDPC})}$ cycles, and the four moves of L_j are moves between zones, each lasting $T_{\text{RESHUFFLE}}^{\text{swap}}(\kappa, 3) = 224$ cycles (Equation (134)). Hence

$$T_{\text{merge}}^{(\text{LDPC})} = 4\,T_{\text{ADDER}}^{(\text{LDPC})}(\kappa{+}1){+}3\,T_{\text{surg}}^{(\text{LDPC})}{+}4\,T_{\text{RESHUFFLE}}^{\text{swap}}(\kappa, 3), \tag{166}$$

that is 6656 cycles. Equation (161) applies with the

durations of Table 27. One erasure lasts $t_{\text{acc}} = 20\,878$ cycles, including its parity-CNOT gates, and the final uncomputation of the exponent lasts $T_{\text{end}} = 683\,088$ cycles. The merges and erasures add 2.2 % to the shot at $\rho = 2$ and 4.9 % at $\rho = 7$. We evaluate the volumes of Equations (143) to (146) on the parallel schedule. In the repetition layer, every engine contributes the buffer positions of its current slice, counted at the peak $9n_{\text{buf}}^{(\text{ADDER})}$ during the merges, the erasures and the slices of T_{other}. The global routing $\Gamma_{\text{global}}(\rho)$ follows the layout above. The LDPC layer counts its $k_L(\rho)$ logical qubits during the whole shot. The factory counts are those of Equation (163).

Figure 56 and Table 35 give the results. The per-shot failure probability falls from 4.2 % at $\rho = 1$ to 2.4 % at $\rho = 7$, so the pair of Section C.8.2 stays below the criterion. $\rho = 3$ costs 57 % more qubits than the serial run (35 413 against 22 593) and divides the expected total runtime by 3.0, from 75.3 h to 25.4 h. $\rho = 7$ costs 170 % more qubits and is 6.8 times faster, 11.1 h in total. An engine costs relatively more than on the repetition-cat chip, since the LDPC layer stores the exponent register more compactly. The LDPC-cat chip nevertheless stays smaller, with 61 053 physical qubits at $\rho = 7$ against 105 434 on the repetition-cat chip. On 10 chips with $\rho = 7$, the expected clock time is 1.25 h on 610 530 physical qubits. The scheduling assumptions of Section D.1.2.vii apply to the shared address rows as they apply to the shared lane.

D.2 FFDLP

D.2.1 Balanced configuration

i Parameters. This section estimates the physical resources of FFDLP-512 on the repetition-cat and LDPC-cat architectures. FFDLP uses the per-shot circuit of RSA-2048 with different register sizes. The FFDLP-512 row of Table 5 gives $n = 512$, $s = 7$, $|P| = 591$, $\ell = 14$ and $\kappa = 26$. The discrete logarithm has $m = 80$ bits (column d of Table 5). The windows are $w_1 = 7$, $w_3 = 3$ and $w_4 = 5$. Section A.5 gives an exponent register of $n_e = m + 2\lceil m/s \rceil = 80 + 2 \times 12 = 104$ qubits. The factories are those of Sections C.7 and C.8, one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory and two $|i\rangle$ factories, which set $T_{\text{cad}} = 21$ at the selected distances. These factories are sized for the error budget of RSA-2048, and we keep them for simplicity. Counting Toffoli gates as in Section C.7.2.ii, FFDLP-512 executes 4.70×10^6 Toffoli gates per shot, about 160 times fewer than RSA-2048, so that the factories cause a failure during a shot with probabilities $\epsilon_{|\mathrm{C}_X\mathrm{C}_X X\rangle} = 9.5 \times 10^{-7}$ and $\epsilon_{|i\rangle} = 1.2 \times 10^{-5}$ only. We also still assume a cycle time of 1 µs, a reaction time of 10 µs, $p_Z = 5 \times 10^{-4}$ and no bit flips.

ii Layout. On the repetition-cat architecture, the layout keeps the zones of Figure 50, with $l_{\text{mem}} = 2\ell + \log(n_e) = 28 + 7 = 35$, $l_{\text{comp}} = 3(\kappa + 1) + 1 = 82$ and $l_{\text{buf}} = 9n_{\text{buf}}^{(\text{ADDER})} = 27$. At $d = 11$, Equation (92) gives $T_{\text{buf}} = 2d + 20 = 42$ and would allow two ADDER buffers. The buffer zone, however, also holds the LOOKUP buffers and the LOOKUP and PHASEUP auxiliary qubits (Section C.7.1). For this reason, we keep $n_{\text{buf}}^{(\text{ADDER})} = 3$. The sum $l_{\text{buf}} + l_{\text{comp}} + l_{\text{mem}} = 144$ exceeds n_e. The bottom rectangle therefore holds only the memory, compute and buffer zones. The top rectangle holds the n_e exponent qubits, above the memory and compute zones. Counting the routing zones as in Equation (103) gives $\Gamma_{\text{global}} = 2h_{\text{routing,fac}}\, l_{\text{fac}} + 2 \times 4 \times 144 - 40 + 2d$, that is $1000 + 1152 - 40 + 2d = 2d + 2112$. The 40 subtracted ancillae lie above the exposed top of the routing zone, in the $144 - n_e = 40$ columns not covered by the top rectangle (Section C.7.1). The $n_e = 104$ ancillae above the top rectangle are exposed as well. Adding the 247 repetition-code columns of the two rectangles other than the routing column (Equation (104)) gives

$$\begin{aligned} n_{\text{total}}(d) &= 247 \times 2d + \Gamma_{\text{global}} + n_{\text{fac}} - 104 \\ &= 496\, d + 3880. \end{aligned} \tag{167}$$

On the LDPC-cat architecture, the layout of Figure 53 applies unchanged. Equation (125) gives $l_{\text{exp}} = 52$, $l_{\text{mem}} = 18$ and $l_{\text{comp}} = 41$ (Table 36), hence $l_{\text{work}} = 59$ and $L = 111$. The distances are $d_{\text{rep}} = 11$ and $d_{\text{LDPC}} = 12$, hence $T_{\text{surg}}^{(\text{LDPC})} = d^\star = 12$ (Equation (78)). The code $[6L + 17, 2L, 12]$ of Table 20, i.e. $[683, 222, 12]$, has $H = 6$, $\Delta_{\text{LDPC}} = 17$ and a right staircase of width $p_r = 4$. Hence $n_L = 683$, $k_L = 222$ and $\Gamma_{\text{LDPC}} = 1144$ (Equation (126)). Since Δ_{LDPC} is odd, the left staircase is rounded up to $\Delta_{\text{L}} = \lceil \Delta_{\text{LDPC}}/2 \rceil = 9$ qubits. Equation (128) then gives 6 656 physical qubits. The LDPC layer holds 1144 of them, the access zone 1405, the buffer routing zone 664, the buffer zone 594, the factory routing zone $1000 - 23 = 977$ and the factories 1872. The buffer routing zone has length $l_{\text{routing,buf}} = 83$. The global routing of Equation (130) is $\Gamma_{\text{global}} = 977 + \Gamma_{\text{addr}} + \Gamma_{\text{work}} = 977 + 208 + 1497 = 2682$ and $\Gamma_{\text{QFT}} = 364$ (Equation (127)), so that the identity of Equation (132) reads $3610 + 2682 + 364 = 6\,656$.

iii Results. On the repetition-cat architecture, the criterion of Section C.7.2.ii selects $d = 11$, with 9 336 physical qubits (Equation (167)). With $p_{\text{logical}} = 3.99 \times 10^{-12}$ (Equation (122)) and $\text{Vol}_{\text{logical}}^{(\text{eff})} = 4.1 \times 10^9$, the exact form of Equation (117) gives $\epsilon_{\text{logical}} = 1.61 \times 10^{-2}$, and the factories bring the per-shot failure probability to $\epsilon = 1.62 \times 10^{-2}$. At $d = 9$, with 8 344 physical qubits, $\epsilon \approx 75\%$.

On the LDPC-cat architecture, $p_{\text{logical}}^{(\text{rep})} = 3.99 \times 10^{-12}$ and $p_{\text{logical}}^{(\text{LDPC})} = 5.53 \times 10^{-12}$ (Equation (147)), with $\text{Vol}_{\text{logical}}^{(\text{eff,rep})} = 2.1 \times 10^9$ and $\text{Vol}_{\text{logical}}^{(\text{eff,LDPC})} = 2.8 \times 10^9$. Equations (148) and (149) then give

					Expected total runtime	
ρ	ϵ	Qubits	Time per shot	$\mathbb{E}[R]$	1 chip	10 chips
1	4.16 %	22.6 k	7.84 h	9.61	75.3 h	8.69 h
2	3.17 %	29.0 k	4.01 h	9.51	38.1 h	4.32 h
3	2.82 %	35.4 k	2.69 h	9.47	25.4 h	2.87 h
5	2.55 %	48.2 k	1.63 h	9.45	15.4 h	1.73 h
7	2.44 %	61.1 k	1.18 h	9.44	11.1 h	1.25 h

Table 35: RSA-2048 on the LDPC-cat chip with ρ residue engines sharing one exponent register, after Ref. [17, Sec. VI.B.1], with the sequential modular merge of Section D.1.2.iii, at $p_Z = 5 \times 10^{-4}$ and $(d_{\text{rep}}, d_{\text{LDPC}}) = (13, 16)$ of Section C.8.2 for every ρ. Qubits follow Equation (165), with shared address rows. The times assume that the engines never stall each other. $\mathbb{E}[R]$ and the total runtimes follow Equation (156), and the last column uses 10 identical chips.

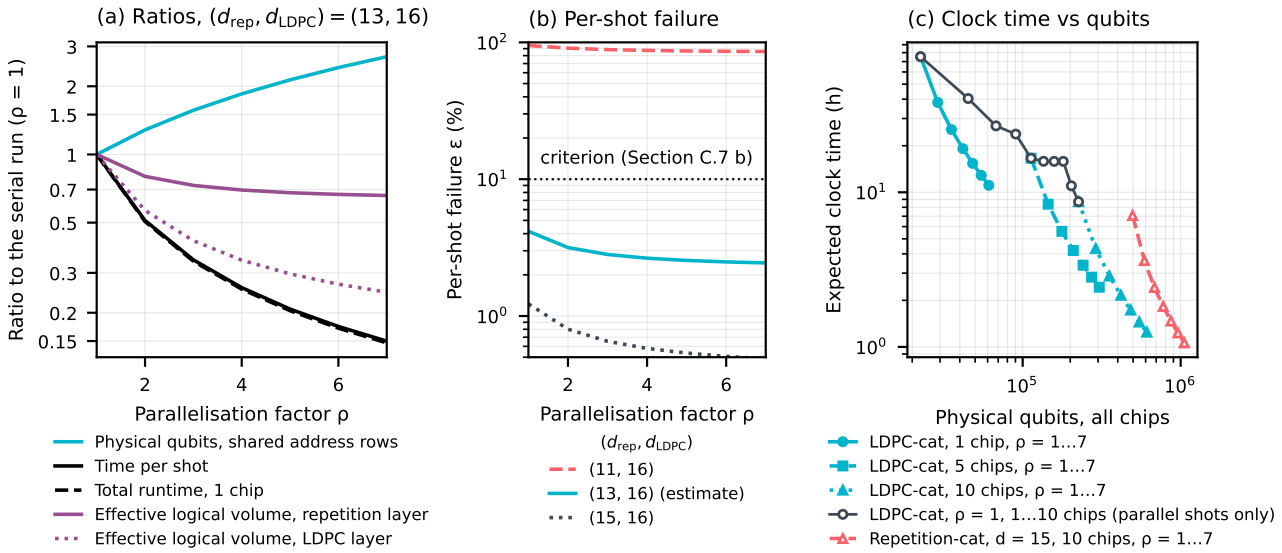


Figure 56: Space-time trade-off for RSA-2048 on the LDPC-cat chip at $p_Z = 5 \times 10^{-4}$, with $\rho \leq 7$ residue engines sharing one exponent register after Ref. [17, Sec. VI.B.1], as in Figure 55. (a) Ratios to the serial run at $(d_{\text{rep}}, d_{\text{LDPC}}) = (13, 16)$, with shared address rows, Equation (165). The effective logical volumes are those of Equations (144) and (146) on the parallel schedule. (b) Per-shot failure probability for three repetition distances at $d_{\text{LDPC}} = 16$, which keep the durations of Table 27. $(13, 16)$ is our estimate. (c) Expected clock time against the total number of physical qubits for one, five and ten LDPC-cat chips, compared with parallel shots alone and with ten repetition-cat chips at $d = 15$, with the shot model of Equation (156). The runtimes assume no routing stalls.

$\epsilon_{\text{logical}}^{(\text{rep})} = 8.21 \times 10^{-3}$, $\epsilon_{\text{logical}}^{(\text{LDPC})} = 1.53 \times 10^{-2}$ and $\epsilon_{\text{total}} = 2.34 \times 10^{-2}$, hence $\epsilon = 2.34 \times 10^{-2}$ with the factories. The criterion selects $(d_{\text{rep}}, d_{\text{LDPC}}) = (11, 12)$ (Table 37), with 6 656 physical qubits. The pairs with fewer physical qubits, such as $(9, 12)$, give $\epsilon > 50\%$.

Durations follow from the call counts of Table 15 applied with the parameters above. The $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory sets $T_{\text{cad}} = 21$ at every distance considered. The register moves that Sections C.7.1 and C.8 leave out of T_{shot} are included here. All register moves take 0.67% of a shot on the repetition-cat architecture, and moves between subroutines take 0.60% on the LDPC-cat architecture. One shot takes 1.37×10^8 cycles (2.28 min) on the repetition-cat architecture and 1.51×10^8 cycles (2.51 min) on the LDPC-cat architecture (Table 38).

For $s = 7$, Equation (60) with $P_{\text{deviant}} \approx 1.9\%$, estimated with Equation (58) as for the FFDLP rows of Table 5, gives $\mathbb{E}[R] \approx 8.24$. Accounting for ϵ, the expected number of shots is 8.37 on the repetition-cat architecture and 8.44 on the LDPC-cat architecture, hence a total runtime of 19.1 min and 21.2 min, respectively. The LDPC-cat architecture thus divides the physical qubit count by 1.4 for an 11% longer runtime. One shot involves about 10^{12} physical qubit-cycles on either architecture, so that neglecting bit flips assumes a bit-flip probability well below 10^{-12} per qubit and cycle, against 10^{-15} for RSA-2048 (Section C.1.1).

D.2.2 Space-efficient configuration

The method of Section D.1.1 applies to the balanced configuration of Section D.2.1, on both archi-

		LDPC-layer columns			Buffer zone
Subroutine	Operation	Memory	Compute	Block	positions
ComputeExponent	Lookup$(7, 21)$	13	32	97	18
	Q-Q Adder(21)	13	32	97	27
ReduceExponent	Q-C Adder(15)	17	26	95	27
	Q-Q Adder(14)	17	25	94	27
ComputeResidue	Lookup$(6, 14)$ (loading)	11	34	97	17
	Lookup$(6, 15)$	18	36	106	17
	Q-Q Adder$^\dagger(15)$	18	36	106	27
	Q-Q Adder(14)	18	34	104	27
AccumulateResidue	Lookup$(5, 27)$	18	41	111	16
	Q-Q Adder$^\dagger(27)$	18	41	111	27
	Q-Q Adder(26)	18	41	111	27
	Lookup$(5, 26)$	18	41	111	16
	Q-Q Comparison(26)	18	41	111	27
	Phaseup(5)	18	41	111	17
ComputeResidue†	Lookup$(6, 15)$	18	36	106	17
	Q-Q Adder$^\dagger(15)$	18	36	106	27
	Q-Q Adder(14)	18	34	104	27
	Lookup$(6, 14)$	18	34	104	17
	Q-Q Comparison(14)	18	36	106	27
	Phaseup(6)	18	34	104	20
	Phaseup(6) (unload)	11	13	76	20
Last Phaseup	Phaseup(7)	13	0	65	27

Table 36: Layout of each slice on the LDPC-cat architecture for FFDLP-512, with parameters ($\ell = 14$, $\kappa = 26$, $n_e = 104$, $w_1 = 7$), $n_{\text{buf}}^{(\text{Adder})} = 3$ and $n_{\text{buf}}^{(\text{Lookup})} = 2$. Block adds the $l_{\text{exp}} = 52$ columns of the exponent zone, used during the whole computation. As for RSA-2048, the accumulation register, packed in 13 columns, is parked in the memory zone during ComputeExponent and ReduceExponent, since $13 + 32 = 45 > 41$. The maxima, 18 memory columns, 41 compute columns and $L = 111$ columns of the block, are reached in AccumulateResidue and fix l_{mem}, l_{comp} and L in the global layout. The 27 buffer positions are used by every Adder and by the last Phaseup(7), which fills the buffer zone. The table omits the M_Z, Reshuffle, parity-Cnot and $P_{|+\rangle}$ slices, which never exceed the adjacent Adder or Lookup slices.

tectures. The slower layout of the $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory sets $T_{\text{cad}} = 62$, and Lookup durations follow Equation (155). The factories and their routing take 1444 instead of 2849 physical qubits on the LDPC-cat architecture, and 1446 instead of 2872 on the repetition-cat architecture.

Table 39 compares the two configurations. The criterion again selects $(d_{\text{rep}}, d_{\text{LDPC}}) = (11, 12)$ on the LDPC-cat architecture and $d = 11$ on the repetition-cat architecture. On the LDPC-cat architecture, the physical qubit count decreases by 21%, to 5 251, and one shot lasts 6.69 min. With $\epsilon_{\text{logical}}^{(\text{rep})} = 1.9 \times 10^{-2}$ and $\epsilon_{\text{logical}}^{(\text{LDPC})} = 4.0 \times 10^{-2}$, $\epsilon = 5.8\%$. The expected 8.75 shots give a total runtime of 58.5 min, 2.76 times longer than the balanced configuration. On the repetition-cat architecture, the configuration uses 7 910 physical qubits, 15% fewer, and runs for 56.1 min, 2.94 times longer. In the balanced configuration, the factories and their routing hold 43% of the physical qubits on the LDPC-cat architecture and 31% on the repetition-cat architecture, against 13% for RSA-2048. The qubit reduction is therefore larger than the 6.2% of RSA-2048, for a similar increase in runtime.

D.2.3 Time-efficient configuration

As for RSA-2048 (Section D.1.2), the shots of FFDLP-512 can run in parallel on k identical chips, with the expected clock time of Equation (156) and $s + 1 = 8$ consistent shots. Table 40 gives the results. With $k = 8$ chips, the average duration drops from 19.1 min to 2.9 min on the repetition-cat architecture, and from 21.2 min to 3.3 min on the LDPC-cat architecture, with 1.26 and 1.30 rounds on average. With 10 chips, the expected clock time approaches the duration of a single shot. The clock time reaches 2.3 min on 93 360 physical qubits using the repetition-cat architecture, and 2.6 min on 66 560 physical qubits using the LDPC-cat architecture.

We do not use parallel residue engines for FFDLP-512. Their benefit comes from sharing the exponent register, which holds only 25 % of the physical qubits on the repetition-cat chip and 16 % on the LDPC-cat chip, against 77 % and 71 % for RSA-2048. An extra

d_{rep}	d_{LDPC}	$\text{Vol}_{\text{logical}}^{(\text{eff,rep})}$	$\text{Vol}_{\text{logical}}^{(\text{eff,LDPC})}$	$\epsilon_{\text{logical}}^{(\text{rep})}$	$\epsilon_{\text{logical}}^{(\text{LDPC})}$	ϵ_{total}	ϵ	
				$p_Z = 10^{-3}$				
9	12	3.0×10^9	2.8×10^9	1.00	6.27×10^{-1}	1.00	1.00	
	16	3.6×10^9	2.4×10^9	1.00	2.38×10^{-3}	1.00	1.00	
	22	4.5×10^9	2.1×10^9	1.00	3.73×10^{-6}	1.00	1.00	
11	12	2.1×10^9	2.8×10^9	3.97×10^{-1}	6.27×10^{-1}	7.75×10^{-1}	7.75×10^{-1}	
	16	2.5×10^9	2.4×10^9	4.55×10^{-1}	2.38×10^{-3}	4.56×10^{-1}	4.56×10^{-1}	
	22	3.0×10^9	2.1×10^9	5.26×10^{-1}	3.73×10^{-6}	5.26×10^{-1}	5.26×10^{-1}	
13	12	1.6×10^9	2.9×10^9	1.07×10^{-2}	6.38×10^{-1}	6.42×10^{-1}	6.42×10^{-1}	
	16	1.8×10^9	2.4×10^9	1.24×10^{-2}	2.38×10^{-3}	1.48×10^{-2}	1.49×10^{-2}	◀
	22	2.2×10^9	2.1×10^9	1.53×10^{-2}	3.73×10^{-6}	1.53×10^{-2}	1.54×10^{-2}	
15	12	1.3×10^9	3.0×10^9	2.41×10^{-4}	6.59×10^{-1}	6.59×10^{-1}	6.60×10^{-1}	
	16	1.4×10^9	2.4×10^9	2.64×10^{-4}	2.38×10^{-3}	2.64×10^{-3}	2.75×10^{-3}	
	22	1.7×10^9	2.1×10^9	3.24×10^{-4}	3.73×10^{-6}	3.28×10^{-4}	4.38×10^{-4}	
				$p_Z = 5 \times 10^{-4}$				
9	12	3.0×10^9	2.8×10^9	5.61×10^{-1}	1.53×10^{-2}	5.68×10^{-1}	5.68×10^{-1}	
	16	3.6×10^9	2.4×10^9	6.28×10^{-1}	9.29×10^{-6}	6.28×10^{-1}	6.28×10^{-1}	
	22	4.5×10^9	2.1×10^9	7.04×10^{-1}	1.82×10^{-9}	7.04×10^{-1}	7.04×10^{-1}	
11	12	2.1×10^9	2.8×10^9	8.21×10^{-3}	1.53×10^{-2}	2.34×10^{-2}	2.34×10^{-2}	◀
	16	2.5×10^9	2.4×10^9	9.83×10^{-3}	9.29×10^{-6}	9.84×10^{-3}	9.85×10^{-3}	
	22	3.0×10^9	2.1×10^9	1.21×10^{-2}	1.82×10^{-9}	1.21×10^{-2}	1.21×10^{-2}	
13	12	1.6×10^9	2.9×10^9	8.82×10^{-5}	1.58×10^{-2}	1.58×10^{-2}	1.59×10^{-2}	
	16	1.8×10^9	2.4×10^9	1.02×10^{-4}	9.29×10^{-6}	1.12×10^{-4}	1.25×10^{-4}	
	22	2.2×10^9	2.1×10^9	1.26×10^{-4}	1.82×10^{-9}	1.26×10^{-4}	1.39×10^{-4}	
15	12	1.3×10^9	3.0×10^9	9.94×10^{-7}	1.67×10^{-2}	1.67×10^{-2}	1.67×10^{-2}	
	16	1.4×10^9	2.4×10^9	1.09×10^{-6}	9.29×10^{-6}	1.04×10^{-5}	2.35×10^{-5}	
	22	1.7×10^9	2.1×10^9	1.34×10^{-6}	1.82×10^{-9}	1.34×10^{-6}	1.45×10^{-5}	

Table 37: Effective logical volumes and logical error probabilities per shot for FFDLP-512 on the LDPC-cat architecture, as a function of the distances d_{rep} and d_{LDPC}, for $p_Z = 10^{-3}$ and $p_Z = 5 \times 10^{-4}$. T_{shot} includes the moves between subroutines. ϵ includes the factories, which cause a failure with probabilities $\epsilon_{|\text{C}_X\text{C}_X X\rangle} = 7.0 \times 10^{-5}$ and $\epsilon_{|i\rangle} = 4.0 \times 10^{-5}$ at $p_Z = 10^{-3}$, and 9.5×10^{-7} and 1.2×10^{-5} at $p_Z = 5 \times 10^{-4}$. All pairs use the surgery durations of Equation (78), $T_{\text{cad}} = \max(T_{\text{surg}}^{(\text{LDPC})}, 21)$, that is 21 for $d_{\text{LDPC}} \leq 16$ and 22 for $d_{\text{LDPC}} = 22$ (Equation (123)), and $n_{\text{buf}}^{(\text{Adder})} = 3$. An entry of 1.00 denotes a probability above 0.995. The pairs marked by ◀ are selected as the pairs with the fewest physical qubits among those with $\epsilon < 10\%$ (criterion of Section C.7.2.ii): $(13, 16)$ at $p_Z = 10^{-3}$, with 7 263 physical qubits, and $(11, 12)$ at $p_Z = 5 \times 10^{-4}$, with 6 656 physical qubits.

engine would thus cost almost as much as an extra chip, and parallel shots reach the same clock times without the scheduling assumptions of Section D.1.2.

D.3 ECDLP

D.3.1 Balanced configuration

This section estimates the physical resources of ECDLP on secp256k1 ($n = 256$), on the LDPC-cat architecture only. The circuit is the gate-optimized windowed point addition of [12] (Section B.2). Table 16 gives the circuit parameters, and Tables 17 and 18 the operation counts. One successful run suffices to recover k (Section A.12 and Table 6). The window register of w qubits is recycled (row "Window recycling" of Table 18), so the layout has no exponent zone. The Adder, the Lookup, the Z-basis measurements, the parity-Cnot gates and the preparations take the costs compiled in Section C.8. The other operations are decomposed into these compiled operations, with a duration estimate but no dedicated layout or schedule. The volume counts both layers, excluding the factories but including their routing zone, as allocated during the whole shot. The resulting estimates are therefore approximate. As in Section D.2, we assume a cycle time of 1 µs, a reaction time of 10 µs, $p_Z = 5 \times 10^{-4}$ and no bit flips.

i Operations without a dedicated compilation. The compiled operations take the LDPC-cat durations of Section C.8.1.iv. All the surgeries of this section involve the LDPC layer, so T_{surg} denotes $T_{\text{surg}}^{(\text{LDPC})} = d^\star$ (Equation (78)) throughout. As in Section C.8, operations are written in the color-changed convention of Section C.3. The $\text{C}X(x)$, $M_X(x)$ and $P_{|0\rangle}(x)$ rows of Tables 17 and 18 thus become parity-

Subroutine	Operation	Calls	Cycles/call	Cycles	Share
ComputeExponent	Lookup$(7, 21)$	8 880	2721	2.42×10^7	16.04%
	Q-Q Adder(21)	8 880	882	7.83×10^6	5.20%
	$M_Z(21, 3)$	8 880	96	8.52×10^5	0.57%
	Reshuffle$(21, 3)$	8 880	108	9.59×10^5	0.64%
ReduceExponent	Q-C Adder(15)	4 137	630	2.61×10^6	1.73%
	Q-Q Adder(14)	4 137	588	2.43×10^6	1.62%
	Reshuffle$(15, 3)$	4 137	108	4.47×10^5	0.30%
	Reshuffle$(2, 1)$	4 137	36	1.49×10^5	0.10%
ComputeResidue	Lookup$(6, 14)$ (loading)	591	1302	7.69×10^5	0.51%
	Lookup$(6, 15)$	8 865	1302	1.15×10^7	7.66%
	Q-Q Adder$^\dagger(15)$	8 865	630	5.58×10^6	3.71%
	$M_Z(15, 3)$	8 865	96	8.51×10^5	0.57%
	Q-Q Adder(14)	8 865	588	5.21×10^6	3.46%
	$M_Z(1)$	8 865	12	1.06×10^5	0.07%
	Reshuffle$(14, 3)$	7 092	108	7.66×10^5	0.51%
	Reshuffle$(14, 3)$ (swap)	1 773	120	2.13×10^5	0.14%
	$M_Z(14, 3)$	1 773	96	1.70×10^5	0.11%
AccumulateResidue	Lookup$(5, 27)$	1 773	603	1.07×10^6	0.71%
	Q-Q Adder$^\dagger(27)$	1 773	1134	2.01×10^6	1.33%
	$M_Z(27, 3)$	1 773	96	1.70×10^5	0.11%
	Q-Q Adder(26)	1 773	1092	1.94×10^6	1.29%
	Lookup$(5, 26)$	886.5	603	5.35×10^5	0.35%
	Q-Q Comparison(26)	886.5	1134	1.01×10^6	0.67%
	$M_Z(26, 3)$	886.5	96	8.51×10^4	0.06%
	Phaseup(5)	1 773	141	2.50×10^5	0.17%
	Reshuffle$(26, 3)$	1 773	108	1.91×10^5	0.13%
ComputeResidue†	Lookup$(6, 15)$	17 730	1302	2.31×10^7	15.33%
	Q-Q Adder$^\dagger(15)$	17 730	630	1.12×10^7	7.42%
	$M_Z(15, 3)$	17 730	96	1.70×10^6	1.13%
	Q-Q Adder(14)	17 730	588	1.04×10^7	6.92%
	Lookup$(6, 14)$	8 865	1302	1.15×10^7	7.66%
	Q-Q Comparison(14)	8 865	630	5.58×10^6	3.71%
	$M_Z(14, 3)$	8 865	96	8.51×10^5	0.57%
	Phaseup(6)	17 730	252	4.47×10^6	2.97%
	Reshuffle$(14, 1)$ (swap)	1 773	192	3.40×10^5	0.23%
	Reshuffle$(14, 3)$	15 957	108	1.72×10^6	1.14%
	Reshuffle$(14, 3)$ (swap)	1 773	120	2.13×10^5	0.14%
	Phaseup(6) (unload)	591	252	1.49×10^5	0.10%
ReduceExponent†	Q-C Adder(15)	4 137	630	2.61×10^6	1.73%
	Q-Q Adder(14)	4 137	588	2.43×10^6	1.62%
	Reshuffle$(15, 3)$	4 137	108	4.47×10^5	0.30%
	Reshuffle$(2, 1)$	4 137	36	1.49×10^5	0.10%
Last Phaseup	Phaseup(7)	15	399	5.98×10^3	0.00%

Table 38: Breakdown of cycle counts by operation for FFDLP-512 on the LDPC-cat architecture, for $p_Z = 5 \times 10^{-4}$ and $(d_{\text{rep}}, d_{\text{LDPC}}) = (11, 12)$. The total duration is $T_{\text{shot}} = 1.51 \times 10^8$ cycles. Lattice surgeries involving the LDPC layer last $T_{\text{surg}}^{(\text{LDPC})} = 12$ cycles, loadings of factory outputs $d_{\text{rep}} = 11$ cycles, and Adders run at the cadence $T_{\text{cad}} = 21$(Equations (135) to (137)). The Lookup$(5, 26)$, Q-Q Comparison(26) and $M_Z(26, 3)$ of AccumulateResidue are performed half of the time on average (Table 15), hence their non-integer number of calls. For clarity, the table omits the parity-Cnot and $P_{|+\rangle}$ slices, which contribute 0.60% of T_{shot}. Unlike in Table 27, T_{shot} includes the moves between subroutines, which are not listed. The final inverse quantum Fourier transform (1.1×10^5 cycles, 0.08% of T_{shot}) is not included.

	$\lvert C_X C_X X\rangle$			Physical		T_{shot}				Runtime	
Configuration	layout	T_{cad}	Distances	qubits	Factories	cycles	min	ϵ	Shots	(min)	Qubit-hours
LDPC-cat architecture											
Balanced	third	21	$(11, 12)$	6 656	2849	1.51×10^8	2.51	2.3%	8.44	21.2	2.35×10^3
Space-efficient	second	62	$(11, 12)$	5 251	1444	4.01×10^8	6.69	5.8%	8.75	58.5	5.12×10^3
Repetition-cat architecture											
Balanced	third	21	$d = 11$	9 336	2872	1.37×10^8	2.28	1.6%	8.37	19.1	2.97×10^3
Space-efficient	second	62	$d = 11$	7 910	1446	3.91×10^8	6.51	4.3%	8.61	56.1	7.39×10^3

Table 39: FFDLP-512 at $p_Z = 5 \times 10^{-4}$ on the two architectures, with the third (balanced, Section D.2.1) and the second (space-efficient) layouts of the $\lvert C_X C_X X\rangle$ factory [47, Table III]. The Factories column includes the factory routing zone. The number of shots is $8.24/(1-\epsilon)$ (Equation (60)), and the qubit-hours are the physical qubit count times the total runtime.

	Repetition-cat			LDPC-cat		
k	Qubits	Rounds	Clock time	Qubits	Rounds	Clock time
1	9.3 k	8.37	19.1 min	6.7 k	8.44	21.2 min
2	18.7 k	4.29	9.8 min	13.3 k	4.34	10.9 min
4	37.3 k	2.27	5.2 min	26.6 k	2.31	5.8 min
8	74.7 k	1.26	2.9 min	53.2 k	1.30	3.3 min
10	93.4 k	1.01	2.3 min	66.6 k	1.02	2.6 min

Table 40: FFDLP-512 on k identical chips running shots in parallel, at $p_Z = 5 \times 10^{-4}$, with the distances of Section D.2.1: $d = 11$ on the repetition-cat chip and $(d_{\text{rep}}, d_{\text{LDPC}}) = (11, 12)$ on the LDPC-cat chip. Rounds and clock times follow Equation (156) with $s + 1 = 8$ consistent shots. Classical pre- and post-processing are not included.

CNOT gates $\mathrm{PC}X(x)$, Z-basis measurements $M_Z(x)$ and $|+\rangle$ preparations $P_{|+\rangle}(x)$. An ADDER and a LOOKUP last $T^{(\text{LDPC})}_{\text{ADDER}}$ and $T^{(\text{LDPC})}_{\text{LOOKUP}}$ (Equations (135) and (136)), and $\mathrm{PC}X(x)$ and $P_{|+\rangle}(x)$ last T_{surg}. A Q-C ADDER is costed as a Q-Q ADDER of the same size, and a COMPARISON of size m as an $\text{ADDER}(m+1)$. Every $M_Z(x)$ is costed $T_{M_Z}(x, 1)$, an upper bound since $T_{M_Z}(x, v) \le T_{M_Z}(x, 1)$ (Equation (133)). We assume that each Toffoli or AND gate outside the compiled operations is limited by the production of its resource states, and lasts one cadence step

$$T_{\text{cad}} = \max\Big(T_{\text{surg}},\, T^{|\mathrm{C}_X\mathrm{C}_X X\rangle}_{\text{prod}},\, T^{|i\rangle}_{\text{prod}}\Big)$$

(Equation (123)). This duration includes the CNOT gate that copies the AND output into an existing target. We assume that such a gate consumes at most one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ state and four $|i\rangle$ states, as a Toffoli gate of the ADDER over its injection and correction steps (Section C.5.4). We neglect the catalyst seeds of isolated gates. An AND gate is uncomputed by measurement. Its target is measured in the X basis and, after the reaction time, a CZ correction is applied to its controls. Each uncomputation therefore lasts about $2T_{\text{surg}} + T_{\text{react}}$.

Controlled Q-Q ADDER. A controlled ADDER maps $|c\rangle\,|x\rangle\,|y\rangle$ to $|c\rangle\,|x\rangle\,|y + c\,x \bmod 2^n\rangle$. We implement the controlled ADDER with two uncontrolled ADDERs and two multi-target CNOT gates, and no other non-Clifford gate. The construction is the single-term case of the controlled add-subtract method of [116]. Two qubits in $|0\rangle$ extend both registers to $n+1$ qubits. The first qubit lies below the least significant qubit of y, so that the extended register holds $Y = 2y$. The second qubit lies above the most significant qubit of x. A first ADDER computes $Y \leftarrow Y + x$. A multi-target CNOT gate $\mathrm{C}X(n+1)$ (Section C.2.2) controlled by c then complements Y when $c = 1$. An $\text{ADDER}^\dagger$ subtracts x, and a second $\mathrm{C}X(n+1)$ undoes the complement. The bitwise complement maps Y to $-Y-1 \bmod 2^{n+1}$, so the conjugated subtraction acts as $Y \leftarrow Y - x$ if $c = 0$ and as $Y \leftarrow Y + x$ if $c = 1$. The extended register therefore ends in $Y = 2(y + c\,x) \bmod 2^{n+1}$. Its n most significant qubits hold $y + c\,x \bmod 2^n$, and both additional qubits return to $|0\rangle$. A controlled n-bit ADDER therefore runs on the layout, buffers and factories of an $(n+1)$-bit ADDER, and consumes twice the resource states of an $(n+1)$-bit ADDER. On the LDPC-cat architecture, the controlled ADDER occupies the $3(n+1)$ slots of an $(n+1)$-bit ADDER (Section C.5.3) and the control qubit. Executing the four operations sequentially bounds its duration by

$$T^{\text{ctrl}}_{\text{ADDER}}(n) \le 2\,T_{\text{ADDER}}(n+1) + 2\,T_{\text{surg}}, \qquad (168)$$

where T_{ADDER} is given by Equation (135). The second term corresponds to the duration of the two multi-target CNOT gates. Controlled Q-C ADDERs are costed as controlled Q-Q ADDERs of the same size, which is an overestimate.

Controlled COMPARISON. A controlled COMPAR-

ison maps $|c\rangle|x\rangle|y\rangle|0\rangle$ to $|c\rangle|x\rangle|y\rangle|c\,b\rangle$, with b the comparison bit of x and y. The comparison circuit computes b as the last carry of a subtraction. An And gate controlled by c and b then copies $c\,b$ into an auxiliary logical qubit prepared in $|0\rangle$. The comparison circuit finally uncomputes its carries, b included, so only $c\,b$ remains. Since we cost a Comparison of size m as an Adder$(m+1)$, adding the duration of the And gate gives

$$T^{\text{ctrl}}_{\text{Comparison}}(m) \simeq T_{\text{Adder}}(m+1) + T_{\text{cad}}. \tag{169}$$

The controlled Comparison occupies the $3(m+1)$ slots of an Adder$(m+1)$, and two slots, for the control and the auxiliary qubit.

CSWAP. A CSWAP on one pair of qubits uses two Cnot gates and one Toffoli gate. With all gates and corrections executed sequentially, the CSWAP of two registers of size n lasts

$$T_{\text{CSWAP}}(n) \simeq n\big[T_{\text{cad}} + 2T_{\text{surg}} + (2T_{\text{surg}} + T_{\text{react}})\big]. \tag{170}$$

MCX. A multi-controlled Not gate with n controls, MC$X(n)$, chains $n-1$ And gates using $n-2$ auxiliary qubits. The auxiliary qubits are then uncomputed by measurement from the last one to the first, since each correction acts on the previous auxiliary qubit. The duration is then about $T_{MCX}(n) \simeq (n-1)\,T_{\text{cad}} + (n-2)(2T_{\text{surg}} + T_{\text{react}})$.

Absorber and Swapper. The Absorber is the in-place compression circuit of [12, Sec. 3.1 and Fig. 1]. The Swapper uses that circuit twice sequentially. The circuit is composed of five Toffoli gates and six Cnot gates. We execute all gates and corrections sequentially, which yields

$$\begin{aligned} T_{\text{Absorber}} &\simeq 5\big[T_{\text{cad}} + (2T_{\text{surg}} + T_{\text{react}})\big] + 6T_{\text{surg}}, \\ T_{\text{Swapper}} &\simeq 2T_{\text{Absorber}}. \end{aligned} \tag{171}$$

Phaseup. The Phaseups in ECDLP work on a window of size $w = 16$, whose power products need more storage than the buffer zone offers. Therefore, we store the $N_{\text{And}} = 2^h + 2^l - w - 2$ auxiliary qubits of the power products in free logical slots of the LDPC layer instead of repetition-code columns. The layout below has room for these auxiliary qubits. The duration of the Phaseup remains the approximate one of Equation (137).

ii Layout. An Adder of size m occupies $3m$ slots of the LDPC layer, for its inputs and its output (Section C.5.3). The width at a row of Tables 17 and 18 is therefore the subroutine scope, plus the out-of-scope qubits, plus the physical footprint of the operation. This footprint counts $3m$ slots for an Adder instead of its logical count. As in [12, Sec. 3.2], where the Bézout reconstruction (replay) step dominates the space, the widest point is the Q-Q Adder$(n+1)$ controlled on b_0 of the replay in in-place div and in-place mul, with

$$\begin{aligned} N_{\max} &= (G + a - 3) + w + 1 + 3(n+2) \\ &= G + a + w + 3n + 4 = 1468 \end{aligned} \tag{172}$$

logical qubits. The terms are the subroutine scope $G+a-3$, the w out-of-scope qubits, the control b_0 and the $3(n+2)$ slots of the controlled Q-Q Adder$(n+1)$ of Equation (168). The residual ancilla count a is approximate (Table 16). All other rows stay below $N_{\max}$. In particular, the widest controlled comparison, the Comparison(μ_R) of the replay, needs $(G + 2n + a - 2\mu_R - 2) + w + 1 + 3(\mu_R + 1) + 1 = G + 2n + a + w + \mu_R + 3$ logical qubits, $n - \mu_R + 1 = 217$ below $N_{\max}$. The CSWAP(n) in the replay needs $G + a - 1 + w + 1 + 2n$ logical qubits, $n + 4 = 260$ below $N_{\max}$. The closest are the controlled Q-C Adder(φ_R) of the replay, 111 below $N_{\max}$, the coordinate Adder$(n+1)$, 171 below, and the controlled Adder$^\dagger(\nu_j)$ at $\nu_j = n$, 187 below. The Phaseup rows need $2n + w$ logical qubits plus the N_{And} product slots. For the parameters of Table 16, the Phaseup rows need 1022 logical qubits, which stays below $N_{\max}$.

With two logical qubits per column, the LDPC code has $L = \lceil N_{\max}/2 \rceil$, $k_L = 2L$ and $n_L = HL + \Delta_{\text{LDPC}}$. Table 20 gives H and Δ_{LDPC} at the chosen d_{LDPC}. The layout is that of Figure 53 without the exponent zone, so all logical qubits lie in one work zone of length L. The buffer routing zone, of height $h_{\text{routing,buf}} = 4$, and its proxy row span this whole length, so Adders, Lookups and CSWAPs reach every logical qubit. The buffer routing zone thus has length $l_{\text{routing,buf}} = L + 2 + p_r + 9(n^{(\text{Adder})}_{\text{buf}} - 1)$. The factory zone and the factory routing zone are those of Section C.8.2.i.

The buffer zone holds $9n^{(\text{Adder})}_{\text{buf}}$ repetition-code columns of distance d_{rep}, with $n^{(\text{Adder})}_{\text{buf}}$ from Equation (124). The buffer zone hosts, in turn, the Adder buffers and the Lookup buffers with their auxiliary qubits. We assume that the other operations that need resource states are not limited by the space in the buffer zone. The Lookup needs $6n^{(\text{Lookup})}_{\text{buf}}$ buffer columns and $w-1$ auxiliary qubits. The buffer zone therefore suffices if $6n^{(\text{Lookup})}_{\text{buf}} + w - 1 \le 9n^{(\text{Adder})}_{\text{buf}}$. At the distances selected below, $n^{(\text{Adder})}_{\text{buf}} = 3$, and the condition holds with equality.

Equation (128), with $l_{\text{work}} = L$ and no Γ_{addr} or

Γ_{QFT}, gives

$$\begin{aligned} n_{\mathrm{total}}^{(\mathrm{LDPC})} &= \Gamma_{\mathrm{LDPC}} + \big[(2H-1)L + 2(L+2) + 2\Delta_{\mathrm{L}} \\ &\qquad + (2H-1)\,p_r\big] + 2h_{\mathrm{routing,buf}}\,l_{\mathrm{routing,buf}} \\ &\quad + 2d_{\mathrm{rep}}\,l_{\mathrm{buf}} + 2h_{\mathrm{routing,fac}}\,l_{\mathrm{fac}} \\ &\quad - \max(l_{\mathrm{fac}} - l_{\mathrm{buf}}, 0) + n_{\mathrm{fac}} \\ &= (4H+7)L + 3\Delta_{\mathrm{LDPC}} + (2H+7)\,p_r \\ &\quad + 18\,n_{\mathrm{buf}}^{(\textsc{Adder})} d_{\mathrm{rep}} + 81\,n_{\mathrm{buf}}^{(\textsc{Adder})} - 52 \\ &\quad + (2h_{\mathrm{routing,fac}} - 1)\,l_{\mathrm{fac}} + n_{\mathrm{fac}}. \end{aligned} \tag{173}$$

The second equality uses $\Gamma_{\mathrm{LDPC}} = 2(H-1)L + 2\Delta_{\mathrm{LDPC}}$ (Equation (126)), $\Delta_{\mathrm{L}} = \Delta_{\mathrm{LDPC}}/2$, $h_{\mathrm{routing,buf}} = 4$, $l_{\mathrm{buf}} = 9n_{\mathrm{buf}}^{(\textsc{Adder})}$ and $l_{\mathrm{fac}} \geq l_{\mathrm{buf}}$.

iii Duration, distances and factories. One shot lasts

$$T_{\mathrm{shot}} = \sum_o N_o\,T_o, \tag{174}$$

where the sum runs over the rows of Tables 17 and 18. Row o has N_o calls of duration T_o, from Section C.8.1.iv or the equations of this section, at $d^\star$. The registers u and v of the dialog are truncated at iteration j to

$$\nu_j = \min\big(n,\ \lceil n - \tfrac{1}{2}\log_2(8/3)\,j\rceil + p\big) \tag{175}$$

qubits [12, Sec. 3.1], with p the register padding of Table 16. For secp256k1, $\nu_j = n$ for the first 54 iterations, then ν_j decreases to 10. The durations of Equations (135), (168) and (170) are affine in the size. The N_{it} iterations of the dialog rows $\mathrm{CSWAP}(\nu_j)$ and controlled $\textsc{Adder}^\dagger(\nu_j)$ therefore last as long as N_{it} calls at the mean width

$$\bar{\nu} = \frac{1}{N_{\mathrm{it}}}\sum_{j=0}^{N_{\mathrm{it}}-1}\nu_j \simeq 149,$$

which we use for these rows. The semi-classical inverse quantum Fourier transform on the window registers is not included. Its $N_A w = 448$ rotations, applied by phase kickback as in Equation (142), take about 5×10^5 cycles and 1.1×10^4 Toffoli gates, less than 2×10^{-4} of T_{shot} and of the Toffoli count. Equation (174) then yields 3.26×10^9 cycles, that is 54 min, which we round up to $T_{\mathrm{shot}} = 60$ min to leave slack for the register moves and the QFT. The volumes and error probabilities below use the rounded T_{shot}.

The LDPC layer contributes $\mathrm{Vol}_{\mathrm{logical}}^{(\mathrm{eff,LDPC})} = k_L T_{\mathrm{shot}}/d_{\mathrm{LDPC}}$ (Equation (146)). The repetition layer, factories excepted, contributes $\mathrm{Vol}_{\mathrm{physical}}^{(\mathrm{rep})} \leq (n_{\mathrm{total}}^{(\mathrm{LDPC})} - \Gamma_{\mathrm{LDPC}} - n_{\mathrm{fac}})\,T_{\mathrm{shot}}$, converted by Equation (144). Since the buffer routing zone spans the whole length L, this bound stays close to the actual allocation. Equations (148) to (149) then give the error probabilities.

The factories of Table 19 run at the cadence of Equation (123), as in Section C.8.2.i. The state count $N_{|\mathrm{C}_X\mathrm{C}_X X\rangle}$ is obtained by summing over the rows of Tables 17 and 18. This yields $N_{|\mathrm{C}_X\mathrm{C}_X X\rangle} \simeq 6.26\times10^7$, one $|\mathrm{C}_X\mathrm{C}_X X\rangle$ state per Toffoli gate. We assume at most four $|i\rangle$ states per Toffoli gate, as in the Adder (Section C.5.6), hence $N_{|i\rangle} = 4N_{|\mathrm{C}_X\mathrm{C}_X X\rangle} \simeq 2.50\times10^8$. RSA-2048 executes 7.56×10^8 Toffoli gates per shot (Section C.7.2.ii), about 12 times more, so the factories sized for RSA-2048 suffice for ECDLP-256. As in Section C.7.2.ii, the factories cause a failure with probability

$$p_{\mathrm{fac}} = 1 - \big(1 - N_{|\mathrm{C}_X\mathrm{C}_X X\rangle}\,p_{Z,\mathrm{out}}^{|\mathrm{C}_X\mathrm{C}_X X\rangle}\big)\big(1 - N_{|i\rangle}\,p_{Z,\mathrm{out}}^{|i\rangle}\big), \tag{176}$$

with the output infidelities $p_{Z,\mathrm{out}}^{j}$ of Table 19. As in Section C.8.2.i, the per-shot failure probability is $\epsilon = 1 - (1-\epsilon_{\mathrm{total}})(1-p_{\mathrm{fac}})$. We apply the criterion of Section C.7.2.ii to the pairs $(d_{\mathrm{rep}}, d_{\mathrm{LDPC}})$ with d_{LDPC} from Table 20 and odd d_{rep} from 9 to 19. Among the pairs for which ϵ is below 10%, the criterion chooses the one with the fewest physical qubits $n_{\mathrm{total}}^{(\mathrm{LDPC})}$ (Equation (173)).

A run fails with probability at most $p_{\mathrm{fail}} = \epsilon_{\mathrm{total}} + p_{\mathrm{fac}} + \delta_\times$, with $\delta_\times$ the corruption bound of Table 6. A failed run is detected classically by checking $Q = [k]P$. The expected number of runs is then $1/(1-p_{\mathrm{fail}})$, and the total runtime is $T_{\mathrm{shot}}/(1-p_{\mathrm{fail}})$.

iv Results. The criterion selects $(d_{\mathrm{rep}}, d_{\mathrm{LDPC}}) = (13, 16)$, with $H = 7$ and $\Delta_{\mathrm{LDPC}} = 26$. Hence $L = 734$, $k_L = 1468$, $n_L = 5164$ and $n_{\mathrm{total}}^{(\mathrm{LDPC})} = 29\,588$ physical qubits, against 29 696 for the next pair with $\epsilon < 10\%$, $(15, 16)$. One shot takes 3.26×10^9 cycles (54 min), rounded up to $T_{\mathrm{shot}} = 3.6\times10^9$ cycles (60 min). Then $\epsilon_{\mathrm{logical}}^{(\mathrm{rep})} = 1.2\times10^{-2}$, $\epsilon_{\mathrm{logical}}^{(\mathrm{LDPC})} = 1.3\times10^{-3}$, $\epsilon_{\mathrm{total}} = 1.3\times10^{-2}$, $p_{\mathrm{fac}} = 2.3\times10^{-4}$, $\epsilon = 1.3\times10^{-2}$ and $p_{\mathrm{fail}} = 2.2\times10^{-2}$. The expected number of runs is 1.03, and the total runtime 61.8 min.

D.3.2 Space-efficient configuration

The method of Section D.1.1 applies to the balanced configuration of Section D.3.1. The slower layout of the $|\mathrm{C}_X\mathrm{C}_X X\rangle$ factory sets $T_{\mathrm{cad}} = 62$, and the factories and their routing take 1444 instead of 2849 physical qubits.

The criterion keeps $(d_{\mathrm{rep}}, d_{\mathrm{LDPC}}) = (13, 16)$, with 28 183 physical qubits, 4.7% fewer than the balanced configuration. Equation (174) gives 7.61×10^9 cycles (127 min) instead of 3.26×10^9. As in Section D.3.1, a margin of about 10% on the duration brings T_{shot} to 8.41×10^9 cycles (140 min). Then $\epsilon_{\mathrm{logical}}^{(\mathrm{rep})} = 2.7\times10^{-2}$, $\epsilon_{\mathrm{logical}}^{(\mathrm{LDPC})} = 3.0\times10^{-3}$, $p_{\mathrm{fac}} = 2.3\times10^{-4}$, $\epsilon = 3.0\times10^{-2}$ and $p_{\mathrm{fail}} = 3.8\times10^{-2}$. The expected number of runs is 1.04, and the total runtime is 146 min, 2.36 times longer than the balanced configuration.

D.3.3 Time-efficient configuration

The time-efficient variants of RSA-2048 and FFDLP-512 do not apply directly to ECDLP, which needs a single successful run and does not make use of RNS-type techniques.

References

[1] Peter W. Shor. "Algorithms for quantum computation: Discrete logarithms and factoring". In Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94). Pages 124–134. Los Alamitos, CA (1994). IEEE Computer Society Press.

[2] Peter W. Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". SIAM Journal on Computing **26**, 1484–1509 (1997). arXiv:quant-ph/9508027.

[3] Élie Gouzien and Nicolas Sangouard. "Factoring 2048-bit RSA integers in 177 days with 13 436 qubits and a multimode memory". Physical Review Letters **127**, 140503 (2021). arXiv:2103.06159.

[4] Élie Gouzien, Diego Ruiz, François-Marie Le Régent, Jérémie Guillaud, and Nicolas Sangouard. "Performance analysis of a repetition cat code architecture: Computing 256-bit elliptic curve logarithm in 9 hours with 126 133 cat qubits". Physical Review Letters **131**, 040602 (2023). arXiv:2302.06639.

[5] Madelyn Cain, Qian Xu, Robbie King, Lewis R. B. Picard, Harry Levine, Manuel Endres, John Preskill, Hsin-Yuan Huang, and Dolev Bluvstein. "Shor's algorithm is possible with as few as 10 000 reconfigurable atomic qubits" (2026). arXiv:2603.28627.

[6] Ryan Babbush, Adam Zalcman, Craig Gidney, Michael Broughton, Tanuj Khattar, Hartmut Neven, Thiago Bergamaschi, Justin Drake, and Dan Boneh. "Securing elliptic curve cryptocurrencies against quantum vulnerabilities: Resource estimates and mitigations". PRX Quantum **7**, 031001 (2026). arXiv:2603.28846.

[7] Thomas Häner, Felix Tripier, Jacob Young, Michael Naehrig, Andrii Maksymov, Safwan Alam, Dmitri Maslov, Matthew Parrott, Yvette de Sereville, Jordan Sullivan, Mark Webster, Nicolas Delfosse, John Gamble, and Martin Roetteler. "Computing 256-bit elliptic curve discrete logarithms in 26 days on a fault-tolerant trapped-ion quantum computer with 20 000 qubits" (2026). arXiv:2609.05625.

[8] Martin Ekerå and Johan Håstad. "Quantum algorithms for computing short discrete logarithms and factoring RSA integers". In Tanja Lange and Tsuyoshi Takagi, editors, Post-Quantum Cryptography – PQCrypto 2017. Volume 10346 of Lecture Notes in Computer Science, pages 347–363. Springer (2017). arXiv:1702.00249.

[9] Alexander May and Lars Schlieper. "Quantum period finding is compression robust". IACR Transactions on Symmetric Cryptology **2022**, 183–211 (2022). arXiv:1905.10074.

[10] Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher. "Reducing the number of qubits in quantum factoring". In Advances in Cryptology – CRYPTO 2025, Part II. Volume 16001 of Lecture Notes in Computer Science, pages 384–415. Springer (2025).

[11] Craig Gidney. "How to factor 2048 bit RSA integers with less than a million noisy qubits" (2025). arXiv:2505.15917.

[12] André Schrottenloher. "Optimized point addition circuits for elliptic curve discrete logarithms" (2026). arXiv:2606.02235.

[13] Han Luo, Ziyi Yang, Jingquan Luo, Ziruo Wang, Yuexin Su, Xiaoming Sun, Lvzhou Li, and Tongyang Li. "Quantum algorithm for elliptic curve discrete logarithms with space-efficient point addition" (2026). arXiv:2607.13816.

[14] Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher. "Reducing the number of qubits in quantum discrete logarithms on elliptic curves". In Advances in Cryptology – EUROCRYPT 2026, Part I. Volume 16541 of Lecture Notes in Computer Science, pages 371–401. Springer (2026).

[15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. "Imperfect forward secrecy: How Diffie-Hellman fails in practice". In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS 2015). Pages 5–17. New York, NY (2015). ACM.

[16] Diego Ruiz, Jérémie Guillaud, Anthony Leverrier, Mazyar Mirrahimi, and Christophe Vuillot. "LDPC-cat codes for low-overhead quantum computing in 2D". Nature Communications **16**, 1040 (2025). arXiv:2401.09541.

[17] Paul Webster, Lucas Berent, Omprakash Chandra, Evan T. Hockings, Nouédyn Baspin, Felix Thomsen, Samuel C. Smith, and Lawrence Z. Cohen. "The Pinnacle architecture: Reducing the cost of breaking RSA-2048 to 100 000 physical qubits using quantum LDPC codes" (2026). arXiv:2602.11457.

[18] Zaki Leghtas, Gerhard Kirchmair, Brian Vlastakis, Robert J. Schoelkopf, Michel H. Devoret, and Mazyar Mirrahimi. "Hardware-efficient autonomous quantum memory protection". Physical Review Letters **111**, 120501 (2013). arXiv:1207.0679.

[19] Mazyar Mirrahimi, Zaki Leghtas, Victor V. Albert, Steven Touzard, Robert J. Schoelkopf, Liang Jiang, and Michel H. Devoret. "Dynami-

cally protected cat-qubits: a new paradigm for universal quantum computation". New Journal of Physics **16**, 045014 (2014). arXiv:1312.2017.

[20] Zaki Leghtas, Steven Touzard, Ioan M. Pop, Angela Kou, Brian Vlastakis, Andrei Petrenko, Katrina M. Sliwa, Anirudh Narla, Shyam Shankar, Michael J. Hatridge, Matthew Reagor, Luigi Frunzio, Robert J. Schoelkopf, Mazyar Mirrahimi, and Michel H. Devoret. "Confining the state of light to a quantum manifold by engineered two-photon loss". Science **347**, 853–857 (2015). arXiv:1412.4633.

[21] S. Touzard, A. Grimm, Z. Leghtas, S. O. Mundhada, P. Reinhold, C. Axline, M. Reagor, K. Chou, J. Blumoff, K. M. Sliwa, S. Shankar, L. Frunzio, R. J. Schoelkopf, M. Mirrahimi, and M. H. Devoret. "Coherent oscillations inside a quantum manifold stabilized by dissipation". Physical Review X **8**, 021005 (2018). arXiv:1705.02401.

[22] A. Grimm, N. E. Frattini, S. Puri, S. O. Mundhada, S. Touzard, M. Mirrahimi, S. M. Girvin, S. Shankar, and M. H. Devoret. "Stabilization and operation of a Kerr-cat qubit". Nature **584**, 205 (2020). arXiv:1907.12131.

[23] N. E. Frattini, R. G. Cortiñas, J. Venkatraman, X. Xiao, Q. Su, C. U. Lei, B. J. Chapman, V. R. Joshi, S. M. Girvin, R. J. Schoelkopf, S. Puri, and M. H. Devoret. "Observation of pairwise level degeneracies and the quantum regime of the Arrhenius law in a double-well parametric oscillator". Physical Review X **14**, 031040 (2024). arXiv:2209.03934.

[24] Ahmed Hajr, Bingcheng Qing, Ke Wang, Gerwin Koolstra, others, and Irfan Siddiqi. "High-coherence Kerr-cat qubit in 2D architecture". Physical Review X **14**, 041049 (2024). arXiv:2404.16697.

[25] Craig Gidney and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits". Quantum **5**, 433 (2021). arXiv:1905.09749.

[26] Raphaël Lescanne, Marius Villiers, Théau Peronnin, Alain Sarlette, Matthieu Delbecq, Benjamin Huard, Takis Kontos, Mazyar Mirrahimi, and Zaki Leghtas. "Exponential suppression of bit-flips in a qubit encoded in an oscillator". Nature Physics **16**, 509–513 (2020). arXiv:1907.11729.

[27] Camille Berdou, Anil Murani, Ulysse Réglade, W. C. Smith, Marius Villiers, J. Palomo, M. Rosticher, A. Denis, P. Morfin, Matthieu Delbecq, Takis Kontos, Natalia Pankratova, Felix Rautschke, Théau Peronnin, Lev-Arcady Sellem, Pierre Rouchon, Alain Sarlette, Mazyar Mirrahimi, Philippe Campagne-Ibarcq, Sébastien Jezouin, Raphaël Lescanne, and Zaki Leghtas. "One hundred second bit-flip time in a two-photon dissipative oscillator". PRX Quantum **4**, 020350 (2023). arXiv:2204.09128.

[28] Antoine Marquet, Antoine Essig, Joachim Cohen, Nathanaël Cottet, Anil Murani, Emanuele Albertinale, Simon Dupouy, Audrey Bienfait, Théau Peronnin, Sébastien Jezouin, Raphaël Lescanne, and Benjamin Huard. "Autoparametric resonance extending the bit-flip time of a cat qubit up to 0.3 s". Physical Review X **14**, 021019 (2024). arXiv:2307.06761.

[29] Ulysse Réglade, Adrien Bocquet, Ronan Gautier, Joachim Cohen, Antoine Marquet, Emanuele Albertinale, Natalia Pankratova, Mattis Hallén, Felix Rautschke, Lev-Arcady Sellem, Pierre Rouchon, Alain Sarlette, Mazyar Mirrahimi, Philippe Campagne-Ibarcq, Raphaël Lescanne, Sébastien Jezouin, and Zaki Leghtas. "Quantum control of a cat qubit with bit-flip times exceeding ten seconds". Nature **629**, 778–783 (2024). arXiv:2307.06617.

[30] Harald Putterman, Kyungjoo Noh, Rishi N. Patel, Gregory A. Peairs, Gregory S. MacCabe, others, and Oskar Painter. "Preserving phase coherence and linearity in cat qubits with exponential bit-flip suppression". Physical Review X **15**, 011070 (2025). arXiv:2409.17556.

[31] Rémi Rousseau, Diego Ruiz, Emanuele Albertinale, Pol d'Avezac, Danielius Banys, Ugo Blandin, Nicolas Bourdaud, Giulio Campanaro, Gil Cardoso, Nathanael Cottet, Charlotte Cullip, Samuel Deléglise, Louise Devanz, Adam Devulder, Antoine Essig, Pierre Février, Adrien Gicquel, Élie Gouzien, Antoine Gras, Jérémie Guillaud, Efe Gümüş, Mattis Hallén, Anissa Jacob, Paul Magnard, Antoine Marquet, Salim Miklass, Théau Peronnin, Stéphane Polis, Felix Rautschke, Ulysse Réglade, Julien Roul, Jeremy Stevens, Jeanne Solard, Alexandre Thomas, Jean-Loup Ville, Pierre Wan-Fat, Raphaël Lescanne, Zaki Leghtas, Joachim Cohen, Sébastien Jezouin, and Anil Murani. "Enhancing bosonic qubit protection by squeezing". Nature Communications (2026). arXiv:2502.07892.

[32] Jérémie Guillaud and Mazyar Mirrahimi. "Repetition cat qubits for fault-tolerant quantum computation". Physical Review X **9**, 041053 (2019). arXiv:1904.09474.

[33] François-Marie Le Régent, Camille Berdou, Zaki Leghtas, Jérémie Guillaud, and Mazyar Mirrahimi. "High-performance repetition cat code using fast noisy operations". Quantum **7**, 1198 (2023). arXiv:2212.11927.

[34] Christopher Chamberland, Kyungjoo Noh, Patricio Arrangoiz-Arriola, Earl T. Campbell, Connor T. Hann, Joseph Iverson, Harald Putterman, Thomas C. Bohdanowicz, Steven T. Flammia, Andrew Keller, Gil Refael, John Preskill, Liang Jiang, Amir H. Safavi-Naeini,

Oskar Painter, and Fernando G. S. L. Brandão. "Building a fault-tolerant quantum computer using concatenated cat codes". PRX Quantum **3**, 010329 (2022). arXiv:2012.04108.

[35] Connor T. Hann, Kyungjoo Noh, Harald Putterman, Matthew H. Matheny, Joseph K. Iverson, Michael T. Fang, Christopher Chamberland, Oskar Painter, and Fernando G. S. L. Brandão. "Hybrid cat-transmon architecture for scalable, hardware-efficient quantum error correction". PRX Quantum **6**, 030305 (2025). arXiv:2410.23363.

[36] Harald Putterman, Kyungjoo Noh, Connor T. Hann, Gregory S. MacCabe, Shahriar Aghaeimeibodi, Rishi N. Patel, Menyoung Lee, William M. Jones, Hesam Moradinejad, Roberto Rodriguez, Neha Mahuli, Jefferson Rose, John Clai Owens, Harry Levine, Emma Rosenfeld, Philip Reinhold, Lorenzo Moncelsi, Joshua Ari Alcid, Nasser Alidoust, Patricio Arrangoiz-Arriola, James Barnett, Przemyslaw Bienias, Hugh A. Carson, Cliff Chen, Li Chen, Harutiun Chinkezian, Eric M. Chisholm, Ming-Han Chou, Aashish Clerk, Andrew Clifford, R. Cosmic, Ana Valdes Curiel, Erik Davis, Laura DeLorenzo, J. Mitchell D'Ewart, Art Diky, Nathan D'Souza, Philipp T. Dumitrescu, Shmuel Eisenmann, Essam Elkhouly, Glen Evenbly, Michael T. Fang, Yawen Fang, Matthew J. Fling, Warren Fon, Gabriel Garcia, Alexey V. Gorshkov, Julia A. Grant, Mason J. Gray, Sebastian Grimberg, Arne L. Grimsmo, Arbel Haim, Justin Hand, Yuan He, Mike Hernandez, David Hover, Jimmy S. C. Hung, Matthew Hunt, Joe Iverson, Ignace Jarrige, Jean-Christophe Jaskula, Liang Jiang, Mahmoud Kalaee, Rassul Karabalin, Peter J. Karalekas, Andrew J. Keller, Amirhossein Khalajhedayati, Aleksander Kubica, Hanho Lee, Catherine Leroux, Simon Lieu, Victor Ly, Keven Villegas Madrigal, Guillaume Marcaud, Gavin McCabe, Cody Miles, Ashley Milsted, Joaquin Minguzzi, Anurag Mishra, Biswaroop Mukherjee, Mahdi Naghiloo, Eric Oblepias, Gerson Ortuno, Jason Pagdilao, Nicola Pancotti, Ashley Panduro, JP Paquette, Minje Park, Gregory A. Peairs, David Perello, Eric C. Peterson, Sophia Ponte, John Preskill, Johnson Qiao, Gil Refael, Rachel Resnick, Alex Retzker, Omar A. Reyna, Marc Runyan, Colm A. Ryan, Abdulrahman Sahmoud, Ernesto Sanchez, Rohan Sanil, Krishanu Sankar, Yuki Sato, Thomas Scaffidi, Salome Siavoshi, Prasahnt Sivarajah, Trenton Skogland, Chun-Ju Su, Loren J. Swenson, Stephanie M. Teo, Astrid Tomada, Giacomo Torlai, E. Alex Wollack, Yufeng Ye, Jessica A. Zerrudo, Kailing Zhang, Fernando G. S. L. Brandão, Matthew H. Matheny, and Oskar Painter. "Hardware-efficient quantum error correction via concatenated bosonic qubits". Nature **638**, 927–934 (2025). arXiv:2409.13025.

[37] Peter Shanahan and Diego Ruiz. "Elevator codes: Concatenation for resource-efficient quantum memory under biased noise" (2026). arXiv:2601.10786.

[38] Michael A. Nielsen and Isaac L. Chuang. "Quantum computation and quantum information". Cambridge University Press. Cambridge (2000).

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

[40] André Schrottenloher. "Building quantum circuits with Qarton". Cryptology ePrint Archive, Paper 2026/2191 (2026).

[41] Matthew P. Harrigan, Tanuj Khattar, Charles Yuan, Anurudh Peduri, Noureldin Yosri, Fionn D. Malone, Ryan Babbush, and Nicholas C. Rubin. "Expressing and analyzing quantum algorithms with Qualtran" (2024). arXiv:2409.04643.

[42] Oded Regev. "An efficient quantum factoring algorithm". Journal of the ACM **72**, 10:1–10:13 (2025). arXiv:2308.06572.

[43] Seyoon Ragavan and Vinod Vaikuntanathan. "Space-efficient and noise-robust quantum factoring". In Advances in Cryptology – CRYPTO 2024, Part VI. Volume 14925 of Lecture Notes in Computer Science, pages 107–140. Springer (2024). arXiv:2310.00899.

[44] Gregory D. Kahanamoku-Meyer, Seyoon Ragavan, and Katherine Van Kirk. "Parallel spooky pebbling makes Regev factoring more practical". In Advances in Cryptology – EUROCRYPT 2026, Part I. Volume 16541 of Lecture Notes in Computer Science, pages 402–430. Springer (2026). arXiv:2510.08432.

[45] Katherine Van Kirk and Craig Gidney. "Streaming Regev's quantum factoring algorithm" (2026). arXiv:2610.06733.

[46] Hugo Jacinto, Christophe Vuillot, Vivien Londe, and Élie Gouzien. "Simultaneous lattice surgery". In preparation: "Simultaneous lattice surgery" (2026).

[47] Larissa Kroell, Hugo Jacinto, Victor Hertz, Axel Pappalardo, Adel Ben Moussa, Manuel Rispler, Peter Shanahan, Élie Gouzien, Christophe Vuillot, and Vivien Londe. "Ground-state energy estimation compiled on cat qubits: 7 164 qubits for FeMoco-54 in 23.5 hours" (2026). arXiv:2610.37840.

[48] Anthropic. "Claude Opus 5.5" (2026). Large language model.

[49] A. Yu. Kitaev. "Quantum measurements

and the Abelian stabilizer problem" (1995). arXiv:quant-ph/9511026.

[50] Dan Boneh and Richard J. Lipton. "Quantum cryptanalysis of hidden linear functions". In Don Coppersmith, editor, Advances in Cryptology – CRYPTO '95. Volume 963 of Lecture Notes in Computer Science, pages 424–437. Berlin, Heidelberg (1995). Springer.

[51] Sean Hallgren, Alexander Russell, and Amnon Ta-Shma. "The hidden subgroup problem and quantum computation using group representations". SIAM Journal on Computing **32**, 916–934 (2003).

[52] Martin Ekerå. "On post-processing in the quantum algorithm for computing short discrete logarithms". Designs, Codes and Cryptography **88**, 2313–2335 (2020).

[53] Robert B. Griffiths and Chi-Sheng Niu. "Semiclassical Fourier transform for quantum computation". Physical Review Letters **76**, 3228–3231 (1996). arXiv:quant-ph/9511007.

[54] Michele Mosca and Artur Ekert. "The hidden subgroup problem and eigenvalue estimation on a quantum computer". In Colin P. Williams, editor, Quantum Computing and Quantum Communications (QCQC 1998). Volume 1509 of Lecture Notes in Computer Science, pages 174–188. Springer (1999). arXiv:quant-ph/9903071.

[55] Simon Parker and Martin B. Plenio. "Efficient factorization with a single pure qubit and $\log n$ mixed qubits". Physical Review Letters **85**, 3049–3052 (2000). arXiv:quant-ph/0001066.

[56] Christof Zalka. "Fast versions of Shor's quantum factoring algorithm" (1998). arXiv:quant-ph/9806084.

[57] Michele Mosca and Christof Zalka. "Exact quantum Fourier transforms and discrete logarithm algorithms". International Journal of Quantum Information **2**, 91–100 (2004). arXiv:quant-ph/0301093.

[58] Godfrey Harold Hardy and Edward Maitland Wright. "An introduction to the theory of numbers". Oxford University Press. Oxford (2008). 6 edition.

[59] Adrien-Marie Legendre. "Théorie des nombres". Volume 1. Firmin Didot. Paris (1830). 3 edition.

[60] Christof Zalka. "Shor's algorithm with fewer (pure) qubits" (2006). arXiv:quant-ph/0601097.

[61] Craig Gidney. "Approximate encoded permutations and piecewise quantum adders" (2019). arXiv:1905.08488.

[62] Thomas Häner, Samuel Jaques, Michael Naehrig, Martin Roetteler, and Mathias Soeken. "Improved quantum circuits for elliptic curve discrete logarithms". In Jintai Ding and Jean-Pierre Tillich, editors, Post-Quantum Cryptography – PQCrypto 2020. Volume 12100 of Lecture Notes in Computer Science, pages 425–444. Cham (2020). Springer. arXiv:2001.09580.

[63] Craig Gidney. "Windowed quantum arithmetic" (2019). arXiv:1905.07682.

[64] Samuel Jaques and Craig Gidney. "Offloading quantum computation by superposition masking" (2020). arXiv:2008.04577.

[65] Dan Harkins and Dave Carrel. "The Internet key exchange (IKE)". RFC 2409. RFC Editor (1998).

[66] Tero Kivinen and Mika Kojo. "More modular exponential (MODP) Diffie-Hellman groups for Internet key exchange (IKE)". RFC 3526. RFC Editor (2003).

[67] John M. Pollard. "Monte Carlo methods for index computation (mod$p$)". Mathematics of Computation **32**, 918–924 (1978).

[68] Paul C. van Oorschot and Michael J. Wiener. "Parallel collision search with cryptanalytic applications". Journal of Cryptology **12**, 1–28 (1999).

[69] Martin Ekerå. "On the success probability of the quantum algorithm for the short DLP". IACR Communications in Cryptology **3**, 10 (2026). arXiv:2309.01754.

[70] Martin Ekerå. "Revisiting Shor's quantum algorithm for computing general discrete logarithms" (2019). arXiv:1905.09084.

[71] John Proos and Christof Zalka. "Shor's discrete logarithm quantum algorithm for elliptic curves". Quantum Information and Computation **3**, 317–344 (2003). arXiv:quant-ph/0301141.

[72] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin Lauter. "Quantum resource estimates for computing elliptic curve discrete logarithms". In Advances in Cryptology – ASIACRYPT 2017, Part II. Volume 10625 of Lecture Notes in Computer Science, pages 241–270. Springer (2017). arXiv:1706.06752.

[73] Daniel Litinski. "How to compute a 256-bit elliptic curve private key with only 50 million Toffoli gates" (2023). arXiv:2306.08585.

[74] Martin Ekerå. "Quantum algorithms for computing general discrete logarithms and orders with tradeoffs". Journal of Mathematical Cryptology **15**, 359–407 (2021).

[75] Craig Gidney. "Spooky pebble games and irreversible uncomputation" (2019). Blog post.

[76] Niels Kornerup, Jonathan Sadun, and David Soloveichik. "Tight bounds on the spooky pebble game: Recycling qubits with measurements". Quantum **9**, 1636 (2025). arXiv:2110.08973.

[77] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. "Encoding electronic spectra in quantum cir-

cuits with linear T complexity". Physical Review X **8**, 041015 (2018). arXiv:1805.03662.

[78] Burton S. Kaliski, Jr. "The Montgomery inverse and its applications". IEEE Transactions on Computers **44**, 1064–1065 (1995).

[79] Tanuj Khattar, Noah Shutty, Craig Gidney, Adam Zalcman, Noureldin Yosri, Dmitri Maslov, Ryan Babbush, and Stephen P. Jordan. "Verifiable quantum advantage via optimized DQI circuits" (2025). arXiv:2510.10967.

[80] Joost Renes, Craig Costello, and Lejla Batina. "Complete addition formulas for prime order elliptic curves". In Advances in Cryptology – EUROCRYPT 2016, Part I. Volume 9665 of Lecture Notes in Computer Science, pages 403–428. Springer (2016).

[81] Thomas Pornin. "A prime-order group with complete formulas from even-order elliptic curves". IACR Communications in Cryptology **1**, 10 (2024).

[82] David Jao, Dimitar Jetchev, and Ramarathnam Venkatesan. "On the bits of elliptic curve Diffie-Hellman keys". In Progress in Cryptology – INDOCRYPT 2007. Volume 4859 of Lecture Notes in Computer Science, pages 33–47. Springer (2007).

[83] Reza Rezaeian Farashahi and Igor E. Shparlinski. "Pseudorandom bits from points on elliptic curves". IEEE Transactions on Information Theory **58**, 1242–1247 (2012). arXiv:1005.4771.

[84] Jeffrey O. Shallit and Jonathan Sorenson. "A binary algorithm for the Jacobi symbol". ACM SIGSAM Bulletin **27**, 4–11 (1993).

[85] Hyunji Kim, Kyungbae Jang, Siyi Wang, Vikas Srivastava, Anubhab Baksi, Gyeongju Song, Hwajeong Seo, and Anupam Chattopadhyay. "New quantum circuits for ECDLP: Breaking prime elliptic curve cryptography". Cryptology ePrint Archive, Paper 2026/106 (2026).

[86] Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher. "Code for "Reducing the number of qubits in quantum factoring"" (2024). Inria CAPSULE team, accessed 2026-10-15.

[87] André Schrottenloher et al. "Qarton 1.0.1" (2026). GitLab repository, accessed 2026-08-18.

[88] André Schrottenloher. "https://gitlab.inria.fr/capsule/qarton-projects/compressed-shor" (2026).

[89] Axel Pappalardo. "https://github.com/alice-bob-sw/schar-code" (2026).

[90] Dominic W. Berry, Craig Gidney, Mario Motta, Jarrod R. McClean, and Ryan Babbush. "Qubitization of arbitrary basis quantum chemistry leveraging sparsity and low rank factorization". Quantum **3**, 208 (2019). arXiv:1902.02134.

[91] Craig Gidney. "A classical-quantum adder with constant workspace and linear gates" (2025). arXiv:2507.23079.

[92] Alysson Gold, J. P. Paquette, Anna Stockklauser, Matthew J. Reagor, et al. "Entanglement across separate silicon dies in a modular superconducting qubit device". npj Quantum Inf. **7**, 142 (2021). arXiv:2102.13293.

[93] Paul Magnard, Simon Storz, Philipp Kurpiers, others, and Andreas Wallraff. "Microwave quantum link between two superconducting circuits housed in spatially separated cryogenic systems". Phys. Rev. Lett. **125**, 260502 (2020). arXiv:2008.01642.

[94] Neereja M. Sundaresan, Nicholas Cotaj, Yadav Prasad Kandel, Eric Lewandowski, Christian Robert Longton, Yves Martin, Jae woong Nah, Kathryn Pooley, Martin Sandberg, Michael Scanlon, Chi Xiong, Emanuil Sashev Yanev, David Abraham, Abhinandan Antony, Vincent Arena, Nicholas Bornman, Khanh Dang, Oliver Dial, Christopher Guarino, Gabriel Haspil, Jared Hertzberg, Kentaro Heya, Swetha Kamlapurkar, Michael Karschner, George Keefe, Eric Liniger, Laura Luo, Thomas G. McConkey, David Olson, Jason Orcutt, Dmitry Ovchinnikov, Timothy Phung, Sankeerth Rajalingam, Michael Selvanayagam, William Shanks, Dongbing Shao, Abigail Shearrow, Adam Stabile, Jiri Stehlik, Joseph Suttle, Jeng-Bang Yau, and Eric Zhang. "Meter-scale quantum interconnects between large quantum processors" (2026). arXiv:2610.29986.

[95] Hugo Jacinto, Élie Gouzien, and Nicolas Sangouard. "Network requirements for distributed quantum computation". Physical Review Research **8**, 013205 (2026). arXiv:2504.08891.

[96] Joshua Ramette, Josiah Sinclair, Nikolas P. Breuckmann, and Vladan Vuletić. "Fault-tolerant connection of error-corrected qubits with noisy links". npj Quantum Information **10**, 58 (2024). arXiv:2302.01296.

[97] H. Bombin and M. A. Martin-Delgado. "Topological quantum distillation". Physical Review Letters **97**, 180501 (2006). arXiv:quant-ph/0605138.

[98] Diego Ruiz, Jérémie Guillaud, Christophe Vuillot, and Mazyar Mirrahimi. "Unfolded distillation: very low-cost magic state preparation for biased-noise qubits". npj Quantum Information **12**, 53 (2026). arXiv:2507.12511.

[99] Austin G. Fowler and Craig Gidney. "Low overhead quantum computation using lattice surgery" (2019). arXiv:1808.06709.

[100] Dominic Horsman, Austin G. Fowler, Simon Devitt, and Rodney Van Meter. "Surface code quantum computing by lattice surgery".

New Journal of Physics **14**, 123011 (2012). arXiv:1111.4022.

[101] Daniel Litinski. "A game of surface codes: Large-scale quantum computing with lattice surgery". Quantum **3**, 128 (2019). arXiv:1808.02892.

[102] Craig Gidney and Austin G. Fowler. "Flexible layout of surface code computations using AutoCCZ states" (2019). arXiv:1905.08916.

[103] Craig Gidney. "Inplace access to the surface code Y basis". Quantum **8**, 1310 (2024). arXiv:2302.07395.

[104] Craig Gidney. "Halving the cost of quantum addition". Quantum **2**, 74 (2018). arXiv:1709.06648.

[105] Jérémie Guillaud, Joachim Cohen, and Mazyar Mirrahimi. "Quantum computation with cat qubits". SciPost Physics Lecture Notes**72** (2023). arXiv:2203.03222.

[106] Victor V. Albert, Chi Shu, Stefan Krastanov, Chao Shen, Ren-Bao Liu, Zhen-Biao Yang, Robert J. Schoelkopf, Mazyar Mirrahimi, Michel H. Devoret, and Liang Jiang. "Holonomic quantum control with continuous variable systems". Physical Review Letters **116**, 140502 (2016). arXiv:1503.00194.

[107] Shruti Puri, Lucas St-Jean, Jonathan A. Gross, Alexander Grimm, Nicholas E. Frattini, Pavithran S. Iyer, Anirudh Krishna, Steven Touzard, Liang Jiang, Alexandre Blais, Steven T. Flammia, and S. M. Girvin. "Bias-preserving gates with stabilized cat qubits". Science Advances **6**, eaay5901 (2020). arXiv:1905.00450.

[108] Qian Xu, Joseph K. Iverson, Fernando G. S. L. Brandão, and Liang Jiang. "Engineering fast bias-preserving gates on stabilized cat qubits". Physical Review Research **4**, 013082 (2022). arXiv:2105.13908.

[109] Ronan Gautier, Alain Sarlette, and Mazyar Mirrahimi. "Combined dissipative and Hamiltonian confinement of cat qubits". PRX Quantum **3**, 020339 (2022). arXiv:2112.05545.

[110] Ronan Gautier, Mazyar Mirrahimi, and Alain Sarlette. "Designing high-fidelity Zeno gates for dissipative cat qubits". PRX Quantum **4**, 040316 (2023). arXiv:2303.00760.

[111] Yufeng Ye, Connor T. Hann, Kyungjoo Noh, Harald Putterman, Ron Belyansky, Arne L. Grimsmo, and Oskar Painter. "Bias-preserving cat-cat CNOT gate via vacuum-conditional beam-splitter" (2026). arXiv:2607.22852.

[112] Christophe Vuillot, Diego Ruiz, Jérémie Guillaud, and Mazyar Mirrahimi. "Quantum nondemolition measurements as a practical primitive for fault-tolerant computation against biased noise" (2026). arXiv:2605.24262.

[113] Diego Ruiz, Jérémie Guillaud, Christophe Vuillot, and Mazyar Mirrahimi. "Biased-noise qubits: a guide to efficient fault-tolerance using the hierarchy of errors" (2026). arXiv:2607.20143.

[114] Laleh Aghababaie Beni, Oscar Higgott, and Noah Shutty. "Tesseract: A search-based decoder for quantum error correction" (2025). arXiv:2503.10988.

[115] Peter Shanahan. "Asymmetrically concatenated LDPC codes". In preparation.

[116] Daniel Litinski. "Quantum schoolbook multiplication with fewer Toffoli gates" (2024). arXiv:2410.00899.